Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Blaž Kovačič

Digitalizacija diagnostičnih in terapevtskih algoritmov za hitro odločanje ob pacientu

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Luka Šajn
Somentor: prof. dr. Dušan Keber

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Študent naj v okviru diplomske naloge preuči strokovno literaturo o diagnostičnih in terapevtskih algoritmih. Opre naj se predvsem na priročnika prof. dr. Dušana Kebra o notranjih boleznih. V nadaljevanju naj razvije spletno aplikacijo, delujočo tako na računalnikih, kot tudi tablicah in pametnih mobilnih telefonih, namenjeno kreiranju novih in spreminjanju obstoječih medicinskih algoritmov. Aplikacija naj bo delujoč prototip, študent pa naj z njo in s pomočjo zdravnikov poskuša simulirati pospešitev zdravstvenih obravnav, zmanjšanje števila strokovnih napak in izboljšati stroškovno učinkovitost.

Za uspešno izdelavo diplomskega dela se najprej zahvaljujem svojemu mentorju doc. dr. Luki Šajnu, ki je dobrovoljno sprejel mentorstvo in ohranjal dobro voljo, spodbudo in kreativno razmišljanje tudi v primerih, ko sem začasno ostal brez idej o nadaljevanju projekta. Zahvala gre tudi prof. dr. Dušanu Kebru, ki mi je že pred časom predstavil idejo o aplikaciji. Na koncu se zahvaljujem še vsem ostalim – družini in prijateljem, ki so me tekom študija podpirali in mi pomagali vstati tudi, ko sem bil na tleh. Iskrena hvala!

Kazalo

Slovar medicinskih izrazov

medicinski izraz	**pomen**
diagnostični algoritem	grafična predstavitev razpoznavanja bolezni z uporabo odločitvenega drevesa
anamneza	natančen pogovor med zdravnikom in bolnikom, iz čigar navedb poskuša zdravnik izluščiti čim več podatkov, s pomočjo katerih izpopolnjuje diagnostični algoritem
medicinska diagnostika	proces, v katerem zdravnik s pomočjo odkrivanja in analiziranja bolnikovih simptomov in znakov ter s pomočjo diagnostičnih preiskav ugotovi bolezen (diagnozo), ki je odgovorna za bolnikovo stanje
terapevtski algoritem	grafična predstavitev zdravljenja bolezni z uporabo odločitvenega drevesa
medicinska terapevtika	veda, ki razvija znanje za individualizirano predpisovanje zdravljenja (terapije)

Seznam uporabljenih kratic

kratica	angleško	slovensko
DB	Database	podatkovna baza
SQL	Structured Query Language	strukturirani podatkovni poizvedovalni jezik
CSS	Cascading Style Sheets	kaskadne stilske predloge
HTML	HyperText Markup Language	jezik za označevanje besedila namenjen za izdelavo spletnih strani
JS	JavaScript	programski jezik JavaScript
DOM	Document Object Model	objektni model dokumenta
JSON	JavaScript Object Notation	notacija za JavaScript objekte
NPM	Node Package Manager	upravitelj Node paketov
RAM	Random Access Memory	bralno–pisalni pomnilnik

Povzetek

Naslov: Digitalizacija diagnostičnih in terapevtskih algoritmov za hitro odločanje ob pacientu

Avtor: Blaž Kovačič

Hitro in učinkovito diagnosticiranje ter zdravljenje bolezni je ključnega pomena. Temelji na analizi velikega števila simptomov, znakov in laboratorijskih izvidov, kar terja sistematičen pristop.

Kot pripomoček so slovenski zdravniki za področje notranjih bolezni razvili diagnostične in terapevtske algoritme. Ti v tiskani obliki obstajajo že več kot dvajset let in kar kličejo po digitalizaciji, ki bi znatno pospešila sprejemanje odločitev, izboljšala stroškovno učinkovitost, pospešila zdravstvene obravnave in zmanjšala število strokovnih napak.

V okviru diplomske naloge sem razvil spletno aplikacijo „Algomed“, ki omogoča elektronsko pregledovanje pa tudi dopolnjevanje, urejanje in kreiranje medicinskih algoritmov. Za razvoj sem uporabil programsko ogrodje MeteorJS, ki je del programskega okolja Node.js.

Algomed je delujoč prototip aplikacije, ki ga je možno izboljševati in razviti do praktične uporabnosti.

Ključne besede: Medicinski algoritem, medicinska diagnoza, medicinska terapija, spletna aplikacija, Node.js, MeteorJS.

Abstract

Title: Digitization of diagnostic and therapeutic algorithms for fast decision making at the patient–side

Author: Blaž Kovačič

Fast and efficient diagnosis and treatment of patient's disease is crucial. It is based on the analysis of a large number of symptoms, signs and laboratory findings which requires a systematic approach.

A group of Slovenian specialists of internal medicine have developed and published diagnostic and therapeutic algorithms. These have existed for more than twenty years in printed form only which calls for digitization which would significantly speed up the decision-making process, improve cost–effectiveness, speed up medical treatment and decrease number of medical errors.

As a part of the thesis I developed a web application "Algomed" that allows electronic examination and also updating and editing of existing algorithms. For the development of the application I used software framework MeteorJS which is part of Node.js software environment.

Algomed is a working prototype of an application which can be upgraded and developed to its practical use.

Keywords: Medical algorithm, medical diagnosis, medical therapy, web application, Node.js, MeteorJS.

Poglavje 1

Uvod

Hitro, pravilno in učinkovito diagnosticiranje ter zdravljenje bolezni sta ključnega pomena pri zdravstveni obravnavi bolnikov. V nekaterih primerih lahko odločata o življenju bolnika, vedno pa o boljšem ali slabšem izidu bolezni in večjih ali manjših stroških. Temeljita na analizi velikega števila podatkov, ki jih zdravnik zbere o obravnavanem bolniku: simptomov, znakov in najrazličnejših preiskav. Množica podatkov dela procesa postavljanja diagnoze in izbire najprimernejšega zdravljenja zelo kompleksna in zahtevna. V primeru, da se ju zdravnik loteva „na pamet“, zgolj z razmišljanjem ob bolniku, sta podvržena vsem tipičnim pomanjkljivostim človeškega „računalnika“: pomanjkljivemu znanju, pozabljanju, trenutnemu počutju zdravnika, razpoložljivemu času itd.

Šele zadnjih nekaj desetletij zdravniki uporabljajo pripomočke, ki niso podvrženi tem slabostim: medicinske algoritme, s katerimi skušajo standardizirati postopke in ob tem vključiti načela medicine, podprte z dokazi ([1], [2]). Slovenski strokovnjaki so pred več kot dvajsetimi leti napisali dva priročnika, ki temeljita na algoritemskem pristopu ([3], [4]). V času njunega nastanka računalništvo ni bilo toliko razvito, da bi omogočilo njuno preoblikovanje v elektronsko obliko. Presenetljivo pa je, da sta vse do danes ostala samo v tiskani obliki. Čeprav sta za razliko od učbenikov še vedno zelo uporabna neposredno med delom z bolnikom, kar kličeta po digitalizaciji (informati-

zaciji), ki bi omogočila, da bi zdravnik s pomočjo ustrezne strojne opreme z njuno vsebino lahko razpolagal v vsakem trenutku in na kateremkoli delovnem mestu. Nekateri primeri digitalizacije v svetu ([5]) ustrezajo specifičnim zahtevam določenega okolja in so za slovenske algoritme neuporabni.

Namen pričujočega diplomskega dela je razvoj aplikacije, ki bi s pomočjo pametnega telefona, računalnika ali tablice omogočala dostop do določenega algoritma in z njim povezanih podatkov – vključno z dostopom do internetnih baz podatkov. Od te aplikacije pričakujemo še večjo uveljavitev algoritemskega pristopa v diagnostiki in zdravljenju, pospešitev zdravstvenih obravnav, zmanjšanje števila strokovnih napak in večjo stroškovno učinkovitost.

Poglavje 2

Teoretične osnove

2.1 Diagnostika in zdravljenje v medicini

Medicinska diagnostika (prepoznavanje) je proces, v katerem zdravnik s pomočjo odkrivanja in analiziranja bolnikovih simptomov in znakov ter s pomočjo diagnostičnih preiskav ugotovi bolezen (diagnozo), ki je odgovorna za bolnikovo stanje. Simptome razkrije zdravnik s pomočjo natančnega pogovora (anamneze), znake pa s telesnim (fizikalnim) pregledom. Kadar je potrebno, odredi preiskave telesnih tekočin ter različne morfološke in funkcionalne preiskave.

Vmesni rezultat diagnostičnega procesa je kopica podatkov – atributivnih in numeričnih – ki jih mora zdravnik smiselno interpretirati, da lahko med številnimi možnostmi določi najverjetnejšo diagnozo. Marsikdaj je to zelo zahtevno. Nekateri simptomi so nespecifični, saj se z njimi kažejo mnoge bolezni (n.pr. utrujenost, glavobol), drugi pa bolj specifični in možnost različnih diagnoz zelo zožijo (n.pr. krvavitev iz telesne odprtine). Vedno pa je odvisno od logičnosti razmišljanja zdravnika in spominskih podatkov, ki jih lahko v določenem trenutku prikliče v svoje razmišljanje, da pride do prave diagnoze.

Miselni proces poteka na principu izločanja manj verjetnih možnosti in vse večjega oženja seznama možnih diagnoz do točke, ko ostane samo ena. Če to počne zdravnik brez pripomočkov, samo na podlagi znanja, ki ga lahko

v določenem trenutku, ko stoji ob bolniku, prikliče v zavest, lahko naredi več napak, ki ga odvedejo vstran od prave diagnoze. Najpogostejša napaka je, da zdravnik na določeno diagnozo sploh ne pomisli (slepa pega). Bolnika bo sicer ustrezno preiskoval in naročil preiskave, ki bodo podprle ali izločile diagnoze, o katerih razmišlja, ne bo pa raziskoval tudi v smer, na katero je pozabil in utegne se zgoditi, da vsaj v začetni diagnostiki ne bo prišel do prave diagnoze. Podobne težave lahko nastopijo, ko se zdravnik, potem ko je prepoznal bolnikovo bolezen, odloča za zdravljenje.

Tudi pri isti bolezni lahko zdravljenje dveh bolnikov poteka zelo različno: odvisno je od resnosti in trajanja bolezni pred zdravljenjem, zapletov bolezni, starosti bolnika, spremljajočih bolezni in številnih drugih okoliščin. Za praktično vsako bolezen obstajajo klinične smernice (ki lahko obsegajo več sto strani besedila in tabel), ki zelo podrobno določajo najbolj primerno obliko zdravljenja za vsako od okoliščin, v katerih poteka bolezen. Zdravnik, ki se pri odrejanju zdravljenja zanaša samo na svoj spomin, marsikdaj ne odredi najboljšega možnega zdravljenja. Pri tem je treba dodati, da se načini zdravljenja zaradi razvoja novih zdravil in zdravstvenih postopkov naglo spreminjajo in tudi zdravnik, ki redno spremlja razvoj znanja na svojem ožjem področju, tega ne more spremljati za vsa področja medicine, s katerimi se srečuje. Problem je največji na tistih delovnih mestih v zdravstvu, kjer bolniki prvič vstopajo v zdravstveni sistem in napačne ali prepočasne odločitve o potrebni zdravstveni obravnavi povzročijo največ škode tako za bolnikovo zdravje (in včasih ogrozijo življenja) kot tudi ekonomsko škodo (dražje zdravljenje, daljše čakalne dobe, preobremenjeno zdravstveno osebje).

2.2 Algoritmi v medicini

Kljub težavam pri odločanju, opisanim v prejšnjem podpoglavju, so se tako mednarodni, kot slovenski strokovnjaki na področju medicine do nedavnega zadovoljevali z razmeroma nepreglednim prikazovanjem najnovejšega znanja

v obliki obsežnih učbenikov, monografij o posameznih boleznih ter seveda s preglednimi članki in kliničnimi smernicami, ki vsebujejo navodila za najboljšo strokovno prakso: medicino, podprto z dokazi. Značilnost vse te literature je, da je zelo malo uporabna, kadar gre ob prizadetem bolniku za vprašanje časa. Zdravnik ob prizadetem bolniku ne more pričeti brskati po indeksu učbenika z nekaj tisoč stranmi, ki je urejen po diagnozah (ob tem pa bolnik diagnoze pogosto sploh še nima), simptomi pa so v stvarnem indeksu omenjeni na nekaj deset različnih straneh. Zato se zdravnik ob bolniku, ki prvikrat vstopi v zdravstveni sistem z določenim problemom, najpogosteje omeji na lastno znanje in izkušnje.

Zavedanje, da potrebujeta procesa diagnostike in zdravljenja podporna orodja, ki bi pospešila odločanje in odpravila možnost napake zaradi nezanesljivosti človeškega spomina ali sposobnosti logičnega sklepanja v kritičnem treutku, se je uveljavilo šele proti koncu prejšnjega stoletja. Orodje se imenuje algoritem.

Algoritem predstavlja zaporedje korakov, ki v končnem številu korakov pripelje do iskane rešitve. V računalniški znanosti si grafično algoritme pogosto predstavljamo kot binarno ali na kak drug način razvejana drevesa. Nič drugače ni v medicini – grafično algoritmi niso nič drugega kot razvejana drevesa z začetnimi, odločitvenimi, akcijskimi in končnimi polji, ki nas postopoma prek odločitev vodijo od začetnega problema do končne rešitve. Drevo odločanja je organizirano po posameznem simptomu, znaku, laboratorijskem izvidu ali bolezni in vodi zdravnika do potrjevanja ali izključevanja in predlogov za dodatna vprašanja ali preiskave do vedno manjšega števila možnosti in končno do enega samega najbolj verjetnega rezultata – bodisi diagnoze ali predloga zdravljenja.

Prvi tiskani priročniki, ki so temeljili na algoritemskem pristopu, so se pojavili v devetdesetih letih prejšnjega stoletja ([3], [4]).

2.3 Slovenski algoritmi za prepoznavanje in zdravljenje notranjih bolezni

Dva priročnika slovenskih avtorjev sta časovno sovpadala s pojavom prvih algoritmov v svetu. Leta 1992 je izšel medicinski priročnik z naslovom Zdravljenje notranjih bolezni ([3]), dve leti kasneje pa še Razpoznavanje notranjih bolezni ([4]).

Slike 2.1 – 2.4 prikazujejo primer algoritma za razpoznavanje simptoma „Utrujenost“ s pripadajočimi opisi anamneze, kliničnega pregleda in potrebnih preiskav. Teh opisov je več, kot jih prikazujejo slike, ki služijo zgolj kot okviren prikaz. Vsak algoritem se prične s prikazom nekaj tipičnih kliničnih primerov.

Slike 2.5 – 2.11 prikazujejo primer terapevtskega algoritma za diagnozo „Srčno popuščanje“ s pripadajočimi opisi zdravstvenih primerov, ukrepov, parametrov in opozoril.

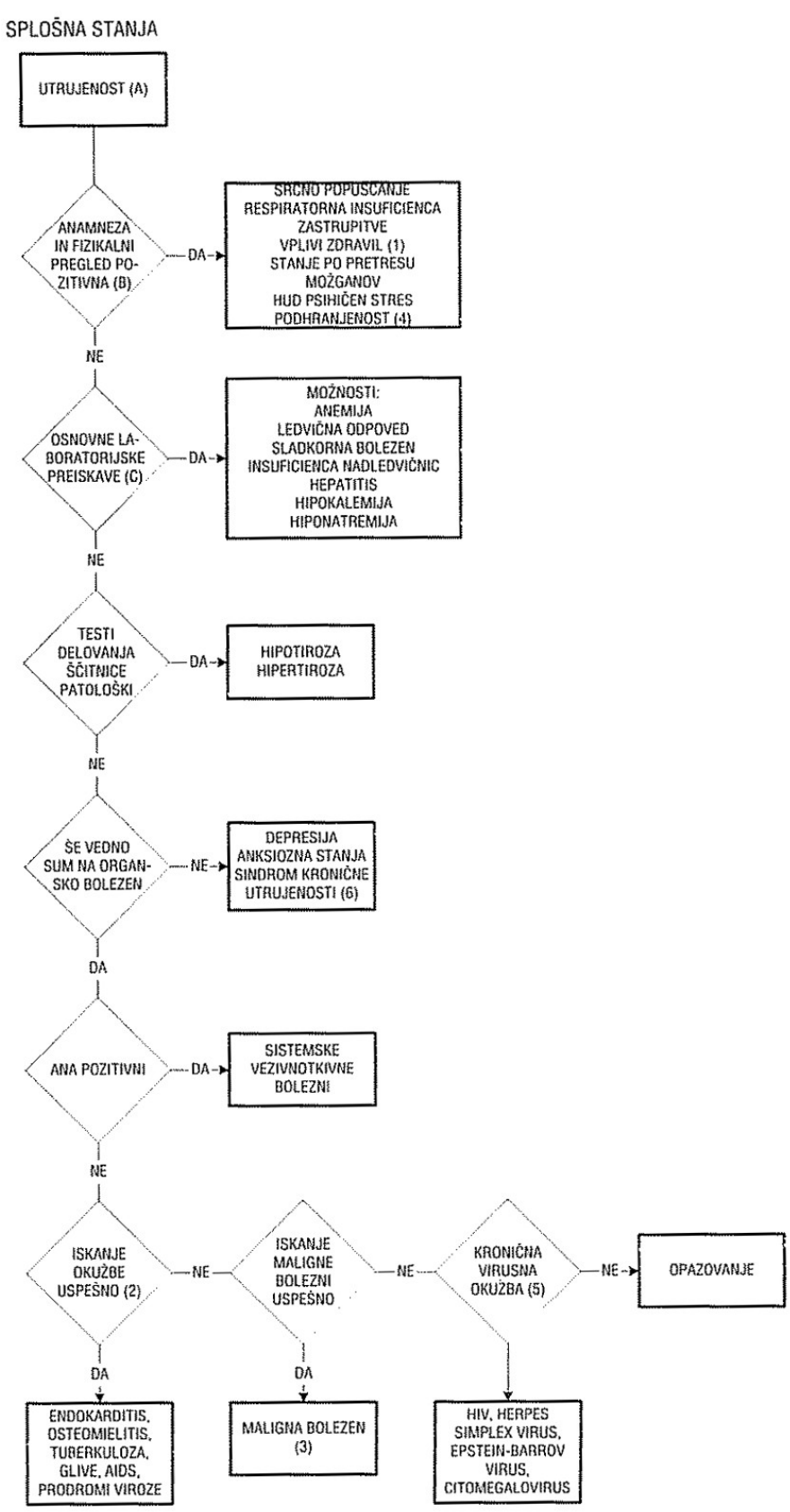
Avtorji si zaradi tiskanega medija niso mogli privoščiti daljših razlag v nekaterih poljih, kot je to vidno že v naslovnem polju, kjer s črko A označujejo, da na strani s podrobnostmi najdemo obsežnejšo razlago za utrujenost. Večina besedila v knjigi razlaga posamezna polja algoritmov.

Sprva sta bila priročnika algoritmov namenjena zdravnikom, kmalu pa so ju začeli uporabljati tudi študenti Medicinske fakultete kot dodatno učno gradivo. Doživela sta izjemno dober sprejem in ju nekateri zdravniki uporabljajo še danes, čeprav sta strokovno, vsaj v svojem terapevtskem delu, precej zastarela.

Algoritmi v obliki tiskanega priročnika na več sto straneh za današnji čas delujejo zelo okorno, saj je njihova uporaba ob bolniku še vedno zamudna (kljub neprimerno hitrejši dostopnosti do ključnih podatkov, kot to omogočajo obsežni medicinski učbeniki, in urejenosti na način, kot poteka razmišljanje ob bolniku).

Da bi se izognili temu še vedno počasnemu „brskanju“ po algoritmu v tiskani obliki, priročnika kar sama kličeta po digitalizaciji in nekemu orodju,

ki bi omogočalo urejanje teh algoritmov ter listanje po njih, s tem pa še hitrejšo diagnostiko in pričetek učinkovitega zdravljenja. V primerjavi s tiskanim priročnikom bi se vsakokratno listanje skrajšalo za več minut, ki so neredko lahko dragocene za izid zdravljenja, vedno pa pomenijo povečanje učinkovitosti delovnega procesa.



Slika 2.1: Izgled algoritma za razpoznavanje bolezni, ki se kaže z utrujenostjo kot vodilnim simptomom.

PRIMER 1

30-letna direktorica tekstilnega podjetja je prišla na pregled zaradi dva meseca trajajoče utrujenosti, ki je bila zjutraj enako močna kot zvečer, čeprav je ponoči spala vsaj osem ur. Vse težje je zmogla delo v službi, zaradi česar je bila močno zaskrbljena. Domača opravila sta skoraj v celoti prevzela njen mož in 9-letna hči. Bolnica si je nekajkrat namerila telesno temperaturo do 37.3° C, ki so jo spremljale rahle bolečine v mišicah in glavoboli v čelnem predelu ali zatilju. Pogosto je imela občutek praskanja v grlu, ki se je po pitju vročega čaja še poslabšalo. Ni kadila. Menstruacije je imela redne. Občasno je vzela kakšno tableto Aspirina ali Brufena, vendar ni opazila izboljšanja počutja. Pri telesnem pregledu je šlo za primerno prehranjeno žensko srednje postave, brez povišane telesne temperature, s krvnim tlakom 115/75 mm Hg. Podkožnih bezgavk nismo tipali. Žrelo je bilo rahlo pordelo z izrazitejšo žilno risbo, nebnice pa niso bile povečane. Fizikalni izvid srca, pljuč in trebuha je bil normalen. Orientacijski nevrološki pregled ni pokazal nikakršnih motenj. Laboratorijske preiskave so pokazale normalen hemogram, SR 8 mm/h, normalne vrednost serumske glukoze, elektrolitov in dušičnih retentov. Koncentracija TSH je bila normalna, antinuklearnih protiteles v serumu ni bilo. EKG in rentgenogram prsnih organov nista pokazala bolezenskih znakov.

Delovna diagnoza: **sindrom kronične utrujenosti.**

PRIMER 2

27-letni študent in aktiven košarkar je prišel na pregled zaradi tri mesece trajajoče, čedalje močnejše utrujenosti, ki se je pojavljala že dopoldne in se je do večera stopnjevala. Treninge je zmogel le s težavo, na tekmah pa je lahko igral največ nekaj minut. Opazil je tudi slabšo koncentracijo pri študiju. Povišane telesne temperature si ni izmeril, shujšal ni. Bil je nekadilec, občasno je užival alkoholne pijače, zdravil ni jemal. Pri telesnem pregledu je šlo za visokega, mišičastega moškega, z normalno telesno temperaturo, s krvnim tlakom 125/80 mm Hg. Desno supraklavikularno smo tipali tri prožne, neboleče, do 1,5 cm velike bezgavke. Fizikalni izvid srca in pljuč je bil normalen. V trebuhu smo tipali rob vranice pod levim rebrnim lokom. Laboratorijske preiskave so pokazale normalen hemogram, SR 32 mm/h, normalne vrednosti serumske glukoze, elektrolitov in dušičnih retentov in normalen TSH. V EKG ni bilo odstopanj od normale, rentgenogram prsnih organov pa je pokazal nakazano razširjen zgornji mediastinum in dvignjen desni pljučni hilus, najverjetneje zaradi povečanih bezgavk.

Delovna diagnoza: **limfadenopatija mediastinalnih in vratnih bezgavk.**

Slika 2.2: Primera dveh bolnikov, kjer se kot vodilni simptom pojavlja utrujenost.

OPOMNIK DIAGNOSTIČNIH POSTOPKOV

(A) Utrujenost. Utrujenost je **subjektivni občutek zmanjšane zmogljivosti za telesno ali duševno delo**. Bolnik jo opisuje tudi kot izčrpanost, pomanjkanje energije, izgubo ambicij ali interesov in zmanjšano vitalnost. Pogosto ga spremlja občutek oslabelosti in intenzivna želja po počitku in spanju. Pri več kot **50 % ljudi**, ki se zatečejo k zdravniku, je utrujenost eden od glavnih ali celo edini simptom.

Utrujenost smatramo za **fiziološko**, če se pojavlja na koncu delovnega dne, po dolgotrajni telesni aktivnosti ali duševnem naporu, po čustvenem stresu ali prekratkem spanju. Kadar pa je utrujenost izrazita, traja več kot le nekaj dni, ovira normalno dnevno aktivnost in je ni mogoče pojasniti s samoumevnimi razlogi, jo je treba raziskati, saj lahko **v ozadju tiči bolezen**. V **20 %** primerov kronične utrujenosti odkrijemo v ozadju **somatski** vzrok, kar pri **80 %** pa so vzroki **psihogeni**.

(B) Anamneza in klinični pregled. Ker je utrujenost v večini primerov **psihogena**, mora anamneza zelo nadrobno razıskati bolnikovo duševno stanje in ıskati znake notranjih nesoglasij. Kadar je utrujenost simptom **depresivnega stanja**, jo spremljajo še drugi znaki depresije (nihanje v razpoloženju čez dan, apatija, pesimizem, žalost, čustveni umik, otopelost, izguba ınteresov ın samospoštovanja), fiziološke motnje (anoreksija, hujšanje, motnje spanja, psihomotorična upočasnitev, zmanjšanje libida oz. ımpotenca) ter kognitivne spremembe (motnje v spomınu in koncentraciji). Več takih simptomov mora zdravnika opozoriti na možnost hude depresije in nevarnost samomora. V takem primeru utegne biti zdravljenje potrebno še pred dokončanjem diagnostike.

Pri **anksioznih stanjih** bolniki tipično povedo, da gredo spat utrujeni in se utrujeni tudi zbudijo. Največkrat se čez dan počutje nekoliko izboljša. Čeprav navajajo stalno utrujenost, pa natančnejša anamneza (ali heteroanamneza) razkrijeta, da se težave lahko v nekaj minutah spremenijo, če je bolnik za določeno dejavnost motiviran. Ta tip utrujenosti pogosto spremljajo glavobol ali druge bolečine ter šibkost.

Utrujenost, ki je posledica **somatske bolezni**, se zmanjša pri zmanjšani aktivnosti, počitku ali spanju. Zjutraj se bolnik zbudi bolj svež, vendar ga običajna dnevna aktivnost utruja.

Pri bolnikih, ki utrujenost zanikajo ali jo vsaj zmanjšujejo, je pa le-ta očitna po njihovem izgledu ali iz pripovedi svojcev, gre pogosteje za organsko bolezen kot za psihogeni vzrok.

Utrujenost se sama po sebi seveda ne ızraža s telesnımi spremembami, ki bi jih odkril **fizikalni pregled**, razen z videzom in obnašanjem. Izraz obraza je pri depresivnih bolnikih velikokrat tipičen. Organske bolezni, ki napredujejo do take stopnje, da pri bolniku povzročijo utrujenost, se pogosto razkrivajo tudi z znaki, ki so dostopni kliničnemu pregledu. Vselej je potreben tudi natančen nevrološki pregled.

(C) Laboratorijske preiskave. Kadar utrujenosti ne spremljajo drugi značilni klinični podatki, ki nas takoj usmerijo k pravi diagnozi, obstajajo pa nespecifične klinične spremembe, ki nakazujejo organsko bolezen v ozadju, je diagnostični postopek odvisen od ocene statistične verjetnosti, da pri posamezniku določenega spola in starosti ter živečem v določenem okolju obstaja določena bolezen. Ob spremenjenih epidemioloških okoliščinah se spreminja verjetnost določenih bolezni. Zato se vrstni red preiskav lahko sorazmerno naglo spremeni.

Poglavitni cilj diagnostične obdelave je izključitev organske bolezni. Preiskave, ki jih smiselno vključujemo v diagnostično obdelavo, so naslednje: hemogram, sedimentacija, preiskava seča, osnovne biokemične preiskave, ščitnični funkcijski testi, glukozni tolerančni test (OGTT), radiogram prsnih organov, elektrokardiogram, presejalni test za sistemske bolezni veziva (določanje prisotnosti in titra avtoprotiteles proti celično nespecifičnim jedrnim antigenom: ANA) serološke preiskave na HIV (če obstaja indikacija). Če so te začetne preiskave negativne, drugih znakov bolezni, razen utrujenosti, pa ni, niso potrebne nadaljnje preiskave, preiskovanec pa naj ostane pod nadzorom, dokler organska bolezen ni dokončno izključena ali pa se pojavijo drugi simptomi in znaki, ki jo razkrijejo. Bolniki, ki so utrujeni zaradi bolezni organskega izvora, bodo prej ko slej poročali o novih simptomih in znakih, ki bodo narekovali dodatno diagnostiko.

Slika 2.3: Del podrobnosti za določena polja algoritma bolezni, ki se kaže z utrujenostjo kot vodilnim simptomom.

DIFERENCIALNODIAGNOSTIČNI OPOMNIK

(1) Vpliv zdravil. Zdravila lahko povzročijo utrujenost z neposrednim učinkom ali pa tako, da spremenijo normalne vzorce spanja ali metabolizem. Taka zdravila so sedativi, hipnotiki, anksiolitiki, nekateri antihistaminiki, večina analgetikov, tudi nesteroidni antirevmatiki, tetraciklini, steroidi, kontracepcijska sredstva, centralno delujoči antihipertenzivi in večina zaviralcev beta. Tudi digitalisova toksičnost se lahko kaže s povečano utrujenostjo. Mnogi hipnotiki, anksiolitiki in triciklični antidepresivi motijo normalno REM spanje. Posledica je nenormalen vzorec spanja, ki vodi v utrujenost, pa čeprav je trajanje spanja normalno.

(2) Infekcijske bolezni. Na akutno bakterijsko bolezen največkrat pomislimo že na osnovi anamneze in fizikalnega pregleda. Kronične okužbe so lahko mnogo bolj prikrite. Raziskati je treba vsak simptom, ki bi lahko pomenil okužbo z bakterijami, glivami, virusi ali bacili tuberkuloze. To zahteva kulture krvi, drugih telesnih tekočin in izločkov. Obvezen je radiogram prsnih organov. Slikanje boleče kosti lahko razkrije osteomielitis. Pri bolnikih, ki navajajo kakršne koli spremembe v odvajanju blata, je potrebna preiskava blata na parazite. Odvisno od spremljajočih simptomov se je včasih treba odločiti za scintigram skeleta ali CT- oz. NMR preiskave različnih delov telesa in organov. Ker utrujenost lahko pomeni enega od začetnih simptomov AIDS-a, je v določeni fazi diagnostičnega postopka, odvisno od anamneze in epidemioloških okoliščin, treba opraviti laboratorijske preiskave na to bolezen.

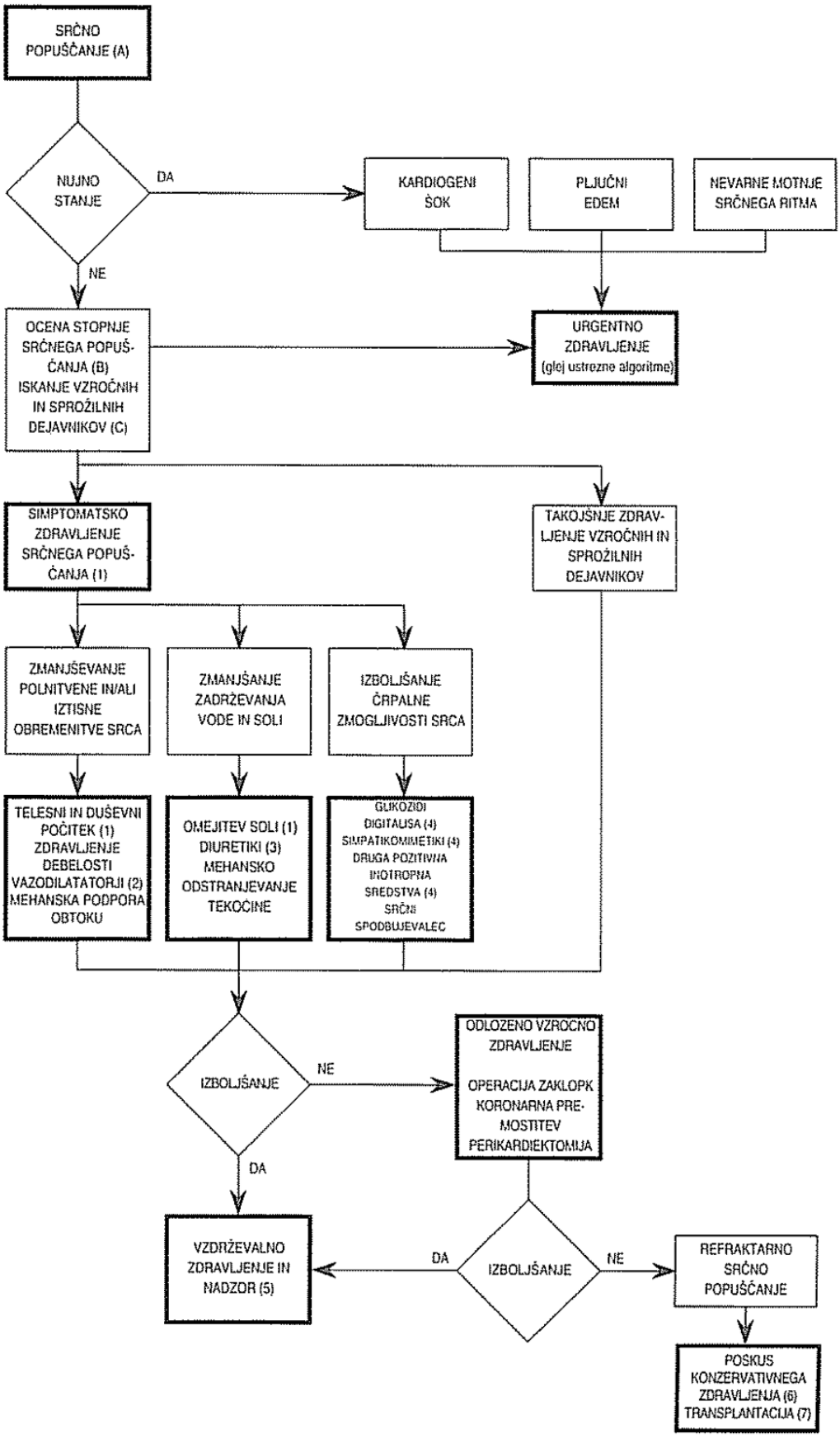
(3) Maligna bolezen. Utrujenost lahko spremlja katero koli maligno bolezen, zlasti v napredovalem stanju. V ta vzrok nas usmerjajo spremembe, ki jih odkrijemo z anamnezo in fizikalnim pregledom, oceniti moramo tudi verjetnost, da pri bolniku obstaja določena maligna bolezen. Utrujenost je posebej značilna za levkemijo in limfome.

(4) Podhranjenost. Pomanjkanje različnih sestavin v prehrani lahko povzroča utrujenost. Najpogosteje gre za pomanjkanje beljakovin, kalorij in nekaterih vitaminov. Čeprav pri nas podhranjenost ni pogost vzrok za utrujenost, jo moramo upoštevati zlasti pri starih in osamelih osebah, ki izgubijo interes za redno prehrano. Misliti je treba tudi na malabsorpcijski sindrom, bizarne prehrambene navade in dolgotrajne shujševalne diete.

(5) Kronične virusne okužbe. Kronične virusne okužbe so kot povzročitelji utrujenosti še nekoliko sporne. Mnogi virusi ostanejo po začetni bolezni prisotni v latentni obliki različnih tkivih: herpes simplex virus, Epstein-Barrov virus in citomegalovirus. Pri bolnikih z zmanjšano imunsko odpornostjo se ti virusi lahko reaktivirajo in povzročijo težke bolezni. Zlasti Epstein-Barrov virus so povezovali z različnimi simptomi, vključno z utrujenostjo. Ni podatkov, da bi kronične virusne bolezni povzročale utrujenost pri posameznikih z normalno imunsko odpornostjo.

(6) Sindrom kronične utrujenosti. Ta sindrom neznane etiologije sestoji iz kronične hude utrujenosti in blagih sistemskih težav v obliki majhnega zvečanja temperature, faringitisa, mišične šibkosti in bolečin v mišicah, bolečih vratnih in pazdušnih bezgavk, glavobola, selečih se bolečin v sklepih ali nevropsihiatričnih motenj, ki jih najdemo pri psihogenih vzrokih utrujenosti. Bolezen v treh četrtinah primerov prizadene ženske, najpogosteje med 20. in 45. letom. Običajno je povezana z aktivnim, zelo stresnim življenjem. Donedavno so jo povezovali s kronično infekcijsko mononukleozo zaradi Epstein-Barrovega virusa, zdaj pa se zdi verjetneje, da gre za kombinacijo stresa, neobičajne oblike depresije in morda alergije. Rutinske laboratorijske preiskave so običajno normalne, zdravljenje pa nezadovoljivo.

Slika 2.4: Del podrobnosti za določena polja algoritma bolezni, ki se kaže z utrujenostjo kot vodilnim simptomom.



Slika 2.5: Izgled algoritma za zdravljenje srčnega popuščanja.

PRIMER 1

40-letni gradbeni delavec je pri osmih letih prebolel revmatično vročino. Kasneje je 10 let jemal penicilin. Težav ni imel, rekreativno se je ukvarjal z nogometom. Sprejet je bil na vroč poletni dan, ko je na delovišču popil nekaj litrov tekočin med intenzivnim delom. Proti koncu delovnega dne je pričel vse težje dihati, tako da je moral počivati, vendar težko dihanje ni povsem ponehalo. Pripeljali so ga na pregled. Ob prihodu se mu je stanje že izboljšalo. Ob pregledu je bil neprizadet, evpnoičen, RR 120/75 mm Hg, nad pljučnimi osnovnicami so bili obojestransko slišni kasni inspiratorni poki. EKG: sinusni ritem, 92/min, nepreprični znaki obremenitve levega preddvora. Rentgenogram prsnih organov: srce normalne velikosti, redistribucija krvnega obtoka v pljučnem žilju. Nad srcem je bil slišen bobneč prvi ton in odpiralni ton mitralne zaklopke. Ultrazvočna preiskava srca je pokazala znake lahke do zmerne mitralne stenoze, ocenjena površina mitralnega ustja 2,4 cm^2.

Diagnoza: **Stenoza mitralne zaklopke,** funkcijski razred I, **akutno levostransko srčno popuščanje** zaradi pretiranega telesnega napora in preobremenitve s tekočinami.

Kako bi bolnika zdravili?

PRIMER 2

62-letni bolnik, dolgoletni hipertonik, je pred 10 leti prebolel spodnjestenski srčni infarkt, pred 3 leti pa še razširjen sprednjestenski infarkt. Od drugega infarkta dalje je jemal Nifecard R 2 × 20 mg dnevno, Moduretic 1 tableto dnevno, Aspirin 100 mg dnevno in Angised lingvalete. Do pred nekaj meseci je bil brez večjih težav in je zmogel zmerne obremenitve. Nato se mu je stanje pričelo postopno slabšati: začel je otekati v noge in v trebuh, ponoči ga je večkrat prebudila težka sapa, pri hoji do prvega nadstropja je moral dvakrat počivati. Zadnje noči je prebedel sede zaradi težke sape. Ob sprejemu je bil bolnik dispnoičen v mirovanju, periferno cianotičen, z nabreklimi vratnimi venami v vsej dolžini v sedečem položaju, testastimi edemi nog do višine kolen, jetra so bila tipna 4 cm pod desnim rebrnim lokom in občutljiva na pritisk, RR 130/80 mm Hg, pulz je bil absolutno aritmičen, 150/min., nad pljuči so bili slišni inspiratorni poki do vrha lopatic, nad srcem je bila tipna patološka sistolična pulzacija prekordija, na konici sta bila slišna III. ton in šum mitralne insuficience. Krvni izvidi: sečnina 17,2 mmol/L, kreatinin 142 μmol/L, K 4,2 mmol/L, Na 145 mmol/L. Rentgenogram prsnih organov: srce izrazito povečano v levo, prerazporeditev krvnega obtoka v pljučnem žilju, kongestija hilusov, intersticijski edem. EKG: atrijska fibrilacija, stanje po srčnem infarktu spodnje in sprednje stene, možnost diskinezije sprednje stene levega prekata. Ultrazvočna preiskava srca: obsežna diskinezija anteroseptalnega predela, dilatacija levega atrija in prekata, funkcijska mitralna insuficienca, iztisni delež 0,20.

Diagnoza: **ishemična bolezen srca, dekompenzirano obojestransko srčno popuščanje**.

Kako bomo bolnika zdravili?

Slika 2.6: Primera dveh bolnikov s srčnim popuščanjem.

DIAGNOSTIČNI OPOMNIK

(A) Srčno popuščanje je patofiziološko stanje, v katerem je zaradi neke nepravilnosti v delovanju srca njegov minutni volumen premajhen za metabolične potrebe perifernih organov ali pa je zadosten za ceno povečanega polnitvenega tlaka. **Simptomi in znaki** srčnega popuščanja so enaki ne glede na vzrok, pač pa so od vzroka odvisni akutnost in zaporedje njihovega pojavljanja ter njihova izraženost.

(B) Ocena stopnje srčnega popuščanja. Uporabljamo različne ocene. Ena je zmogljivost za napor na osnovi anmaneze, pri čemer uporabljamo **funkcijske razrede** newyorške srčne zveze (NYHA) od I do IV. Druga ocena je presoja o tem, ali je bolnik kompenziran (primeren minutni volumen, kar ne izključuje **simptomov in znakov povečanega polnitvenega tlaka** v desnem srcu) ali dekompenziran (simptomi in **znaki premajhnega minutnega volumna**: utrujenost, periferna cianoza, simptomi zmanjšanega pretoka skozi možgane, prerenalna azotemija, itd.). Ob zmanjšanem minutnem volumnu je treba ugotoviti, ali so ob njem prisotni znaki zvečanega ali zmanjšanega polnitvenega tlaka desnega in levega prekata.

(C) Nekatere **vzročne dejavnike** je možno in tudi potrebno odpravljati obenem s simptomatskim zdravljenjem srčnega popuščanja, druge pa lahko odložimo na kasnejši čas. Prav tako je potrebno zdraviti vse **sprožilne dejavnike**. Zato je sestavni del začetne klinične ocene iskanje vseh teh dejavnikov. Med najpomembnejše spadajo pljučna embolija, pljučnica in druge okužbe, anemija, tirotoksikoza, aritmije, revmatični in drugi miokarditisi, akutni in subakutni endokarditis, arterijska hipertenzija, miokardni infarkt in njegovi zapleti (ruptura stene ali papilarne mišice) pretirani telesni ali duševni stresi, opustitev zdravil. Vsi ti dejavniki so lahko sprožilni ob že predhodni srčni bolezni, nekateri pa so lahko vzrok za srčno popuščanje tudi brez poprejšnje bolezni.

Slika 2.7: Prvi del podrobnosti za določena polja algoritma srčnega popuščanja.

TERAPEVTSKI OPOMNIK

(1) Odločitev o načinu **simptomatskega zdravljenja srčnega popuščanja** je odvisna od stopnje srčnega popuščanja, količine pridržane tekočine v zunajceličnem in znotrajžilnem prostoru in stopnje periferne vazokonstrikcije. Zdravljenje je zato usmerjeno v zmanjševanje polnitvene in/ali iztisne obremenitve srca, odstranjevanje odvečne vode in soli ter izboljšanje črpalne zmogljivosti srca. Potrebno je natančno spremljanje enostavnih **kliničnih parametrov**: telesne teže, utripne frekvence, krvnega tlaka, količine zaužitih in izločenih tekočin in klinične ocene venskega tlaka iz vratnih ven. V težjih primerih je potrebno tudi **invazivno merjenje** centralnega venskega tlaka, tlakov v pljučni arteriji in kapilarnega zagozditvenega tlaka. Izbor in intenzivnost zdravljenja sta odvisna od izraženosti posameznih simptomov in znakov. Stopnjevanje posameznih zdravilskih postopkov glede na resnost bolezni prikazuje tabela 1.

Tabela 1. Stopnjevanje posameznih zdravilskih ukrepov pri srčnem popuščanju.

1. OMEJITEV TELESNE AKTIVNOSTI
   a) opustitev zelo napornih športov in težkega telesnega dela
   b) zamenjava delovnega mesta, če je to povezano s telesno aktivnostjo
   c) skrajšanje običajnih dnevnih aktivnosti, počitki čez dan
   č) omejitev na dom
   d) omejitev na stol ali postelljo
2. OMEJITEV UPORABE SOLI
   a) opustitev soljenja pri mizi (Na = 1,6 do 2,8 g)
   b) opustitev soljenja pri kuhanju in mizi (Na = 1, 2 do 1,8 g)
   c) poleg a) in b) še posebna hrana z malo soli (Na = 0,2 do 1,0 g)
3. DIURETIKI
   a) blagi diuretiki (tiazidi)
   b) diuretiki Henlejeve zanke (etakrinična kislina, furosemid)
   c) diuretiki Henlejeve zanke skupaj z diuretikom, ki varčuje kalij
   č) diuretik Henlejeve zanke + tiazid + diuretik, ki varčuje kalij
4. VAZODILATATORJI
   a) ACE inhibitor ali kombinacija hidralazina z isosorbiddinitratom
   b) zvečanje odmerka oralnih vazodilatatorjev
   c) nitroprusid i.v.

Dodatni dejavniki, ki vplivajo na terapevtsko odločitev, so bolnikova starost, poklic, osebnost, življenjski slog, družinske razmere, motivacija za zdravljenje in -nadvse pomembno - odziv na začetno zdravljenje.

Asimptomatski bolniki (funkcijski razred I) ne potrebujejo simptomatskega zdravljenja. Nekatere študije nakazujejo, da uporaba konvertaznega (ACE) inhibitorja že v tem obdobju vsaj pri bolnikih po srčnem infarktu ali z aortno ali mitralno insuficienco izboljša prognozo. Doktrine še ni.

Ob **najzgodnejših simptomih srčnega popuščanja**, ki se običajno pojavijo med velikim telesnim naporom, najpogosteje zadoščajo enostavni ukrepi: omejitev telesne aktivnosti in uporabe soli ter zdravljenje morebitnih poslabševalnih dejavnikov.

Če simptomi in znaki srčnega popuščanja kljub temu vztrajajo ali se ponovno pojavijo, kar je pogosteje pri **bolnikih v II. in III. funkcijskem razredu**, pričnemo zdraviti z diuretikom in vazodilatatorjem, pri čemer se pri slednjem v večini primerov odločimo za ACE inhibitor. Začetni izbor je lahko drugačen pri bolnikih s kronično atrijsko fibrilacijo (če se ne odločimo za konverzijo ritma): v tem primeru pričnemo zdraviti z digitalisom in po potrebi dodamo ACE inhibitor.

Slika 2.8: Drugi del podrobnosti za določena polja algoritma srčnega popuščanja.

Tabela 2. Izbira zdravljenja z ozirom na težino srčnega popuščanja pri ishemični srčni bolezni.

ASIMPTOMATSKI BOLNIK	**Odpravljanje dejavnikov tveganja**
	Opustitev kajenja
	Zdravljenje debelosti
	Zdravljenje hiperlipidemije
	Zdravljenje arterijske hipertenzije
	ACE inhibitor? (po srčnem infarktu, ob aortni ali mitralni regurgitaciji)
SIMPTOMATSKI BOLNIK	**Omejitev telesne aktivnosti**
	Omejitev soljenja
	Zdravljenje z zdravili
	Diuretik
	Vazodilatator
	Digoksin (običajno v kombinaciji)
BOLNIK Z REFRAKTARNIM SRČNIM POPUŠČANJEM	**Intravenska inotropna sredstva in vazodilatatorji**
	Dopamin, dobutamin, amrinon, nitroprusid
	Mehanska podpora obtoku
	Mehansko odstranjevanje tekočin
	Torakocenteza, paracenteza, ultrafiltracij, dializa

(2) Vazodilatatorje delimo glede na mesto delovanja na pretežno venske, pretežno arteriolarne in uravnotežene dilatatorje (tabela 3). Ob težkem srčnem popuščanju (npr. ob pljučnem edemu) uporabljamo parenteralne vazodilatatorje (nitroglicerin ali natrijev nitroprusid), pri čemer je potreben natančen nadzor hemodinamskih parametrov. Zaradi nevarnosti hipotenzije začnemo z majhnim odmerkom in jih tudi čimprej opustimo (takoj, ko pade kapilarni zagozditveni tlak v pljučih pod 15 mm Hg in srčni indeks poraste na 2 litra/min/m^2). Lahko so učinkoviti tudi pri kroničnem refraktarnem srčnem popuščanju.

Oralni vazodilatatorji. Vse bolj prevladuje stališče, naj vsi bolniki s kroničnim srčnim popuščanjem zaradi okvare levega prekata in brez kontraindikacije (hipotenzija, ledvična odpoved) prejemajo ta zdravila. Nedvomno je ustrezneje uporabiti ta zdravila kot pa najstrožje omejiti uporabo soli in uvesti kombinirano diuretično zdravljenje ali intravenska inotropna sredstva. Izbiramo predvsem med ACE inhibitorji, nitrati in hidralazinom. **Nitrati** kot izraziti venodilatatorji prihajajo v poštev pri bolnikih z

Tabela 3. Mesto delovanja in odmerki vazodilatatorjev (nitrati kot predstavniki venodilatatorjev so prikazani v tabeli 2 v poglavju Kronična stabilna angina pektoris).

Zdravilo	Mehanizem delovanja	Venodilatacija	Arteriolarna dilatacija	Običajni odmerek
nitroprusid (Nipride)	Direktno	+++	+++	5-150 μg/min i.v.
hidralazin (Ileton)	Direktno	–	+++	10-100 mg 4×/d p.o.
prazosin (Vasoflex)	Blokator α	+++	+	1-5 mg 4×/d p.o.
doksazosin (Tonocardin)	Blokator α	+++	+	4-16 mg 1×/d p.o.
kaptopril (Zorkaptil)	ACE inhibitor	+	+	6,25-25,0 mg 3-4×/d p.o.
enalapril (Olivin, Enap)	ACE inhibitor	+	+	2,5-15 mg 2×/d p.o.
lizinopril (Irumed)	ACE inhibitor	+	+	5-40 mg 1×/d p.o.

Slika 2.9: Tretji del podrobnosti za določena polja algoritma srčnega popuščanja.

PRIMERI ZDRAVLJENJA

PRIMER 1

- Opustitev napornih telesnih aktivnosti in razporeditev na delovno mesto z majhnimi telesnimi obremenitvami.
- Opustitev soljenja pri mizi.
- Navodilo o preprečevanju bakterijskega endokarditisa.
- Enkrat letno kontrola pri kardiologu in pregled srca z ultrazvokom.

PRIMER 2

- Sprejem v bolnišnico, počitek v postelji in omejitev soli.
- Opustitev Nifecarda R (zaradi negativnega inotropnega učinka).
- Furosemid (Lasix) 40 mg i.v.
- Enalapril (Olivin) 2 × 2,5 mg prvi dan, nato ob odsotnosti sopojavov zvečanje na 2 × 5 mg dnevno do največ 2 × 15 mg dnevno.
- Lanitop v vzdrževalnem odmerku 0,2 mg i.v. (alternativa: 3 × 0,2 mg prvi dan, nato 0,2 mg dnevno).
- Aspirin 100 mg dnevno, heparin v preventivem odmerku 5000 E s.c. na 12 ur ali nizkomolekularni heparin (Frahepan) 3000 E s.c. na 24 ur.
- Ocenjevanje zdravljenja: če telesna teža ne pada za 0,5 do 1 kg na dan, zvečanje odmerka Lasixa ali dodatek klortalidona (Hygroton) 25 mg vsak drugi dan p.o.
- Odločanje o prehodu na peroralno zdravljenje: odvisno od izboljševanja kliničnega stanja in izginevanja znakov sistemske kongestije.
- Redno spremljanje elektrolitov in dušičnih retentov (2-krat tedensko).
- Predpis vzdrževalnega zdravljenja ob odpustu: ista zdravila v enakem odmerku kot zadnje dni zdravljenja v bolnišnici.
- Pričakovani učinek zdravljenja: povrnitev v III. funkcijski razred.

Slika 2.10: Primeri zdravljenja srčnega popuščanja.

OPOZORILA IN AFORIZMI

- Pri bolniku s srčnim popuščanjem ne uporabljaj fiziološke raztopine (vsaj ne v večjih količinah) kot nosilno raztopino za zdravila. 500 ml raztopine vsebuje 4,5 g NaCl!
- Pri huje oslabljeni sistolični funkciji levega prekata ne uporabljaj kalcijevih antagonistov!
- Izguba telesne teže ob zdravljenju z diuretiki ne sme presegati enega kilograma na dan.
- Hiponatriemija z znaki kongestije je dilucijska; zdravimo jo z omejitvijo vnosa tekočin in ne z raztopino NaCl.
- Pri hipovolemičnem srčnem bolniku so ACE inhibitorji nevarni.
- Preden proglasimo srčno popuščanje za refraktarno, moramo iskati morebitne prezrte sprožilne dejavnike popuščanja in zaplete samega zdravljenja.

Slika 2.11: Opozorila in aforizmi za zdravljenje srčnega popuščanja.

Poglavje 3

Tehnična osnova za razvoj aplikacije

Pri načrtovanju razvoja programske opreme odigra veliko vlogo izbira načina razvoja. Pri razvoju orodja, ki bi odpravilo težave, omenjene v prejšnjem poglavju, sem tudi sam v fazi načrtovanja veliko časa posvetil premisleku o vrsti aplikacije. Vrsti aplikacij sta dve: spletne aplikacije in mobilne aplikacije.

Mobilne aplikacije

Mobilna aplikacija je programska oprema, namenjena za uporabo na mobilnih napravah – pametnih telefonih in tabličnih računalnikih.

Vse mobilne aplikacije so, če le niso del operacijskega sistema, po razvoju umeščene na trg (najznamenitejša sta Google Play in AppStore), uporabniki pa do njih dostopajo tako, da si v primeru zahtevane licence le–to priskrbijo, nato pa aplikacijo prenesejo na svojo napravo.

Posledično to pomeni, da se napravi v trenutku prenosa in namestitve aplikacije zmanjša preostali pomnilniški prostor. Dodatno prostorsko težavo povzročajo naknadne posodobitve in dodatni paketi aplikacije, ki so prav tako kot glavna aplikacija prenešene na napravo, pogosto pa zasedajo celo več prostora kot osnovna programska oprema.

Spletne aplikacije

Spletna aplikacija je, za razliko od mobilne aplikacije, programska oprema, namenjena za uporabo na katerikoli napravi, vendar preko „posrednika“ – spletnega brskalnika.

Ta aplikacija je dostopna na svetovnem spletu bodisi v oblačni storitvi bodisi na osebnem strežniku in je ni potrebno prenašati na odjemalčevo napravo, kar seveda velja tudi za njene posodobitve. To pomeni, da ne porablja pomnilniškega prostora. Spletni brskalnik sicer med rabo aplikacije zasede nekoliko več RAM prostora, ki pa se po prenehanju njene rabe sprosti.

Odkrivanje novih znanj na področju razvoja spletnih aplikacij je razvijalcem omogočil njihovo hitrejšo in enostavnejšo implementacijo, prav tako pa zmanjšanje porabe procesorja in pomnilniškega prostora odjemalčeve naprave.

Zaradi omenjenih razlogov sem se tudi sam odločil za razvoj spletne aplikacije.

Spletne aplikacije so lahko razvite z raznoraznimi programskimi jeziki, ki nemalokrat vsebujejo še množico knjižnic in razvojnih okolij. Za programski jezik in okolje za orodje, ki sem ga razvil v okviru diplomskega dela, sem izbral JavaScript in Node.js. V podpoglavju 3.1 opisujem v zgoščeni obliki programsko okolje, kot je opisano v literaturi ([6]).

3.1 Node.js

Node.js, pogosto okrajšan kot NodeJS ali pa Node, je izraz za JavaScript strežniško okolje, ki temelji na Googlovem V8 JavaScript sistemu.

V8 je Googlov visokozmogljivi odprtokodni sistem, napisan v programskem jeziku C++. Namenjen je prevajanju in izvajanju JavaScript izvorne kode v Googlovem brskalniku „Google Chrome“; skrbi za alokacijo spomina objektov in čiščenje ter obnavljanje pomnilniškega prostora. Lahko teče kot samostojna aplikacija, lahko pa se ga vključi v katerokoli aplikacijo, napisano v programskem jeziku C++.

JavaScript je najbolj razširjen in uporabljen programski jezik za skriptiranje na strani odjemalca in za manipulacijo DOM objektov tako na strani odjemalca kot strežnika. Jezik se je razvil neodvisno od na prvi pogled podobnega programskega jezika Jave, si pa z njo deli številne lastnosti in strukture.

Za razliko od sodobnih razvojnih okolij procesi, ki jih poganja Node, ne temeljijo na večnitnosti, pač pa na asinhronem vhodno-izhodnem modelu. Asinhroni vhodno-izhodni model je oblika vhodno-izhodnega procesiranja, ki dovoljuje druge obdelave, še preden je trenutni vhodno-izhodni prenos končan. To je tudi glavna razlika med Node.js in ostalimi programskimi jeziki ter okolji, saj se ostali po večini pojavljajo v obliki knjižnic, katerih procesi so večinoma večnitni.

Ko govorimo o Node.js, je potrebno omeniti tudi NPM, ki je nepogrešljiv del razvijanja z Node. NPM skrbi za vključevanje zunanjih paketov in knjižnic v projekte, ne da bi bilo razvijalcem potrebno obiskovati spletno stran vsakega paketa, prenesti na napravo čarovnika za namestitev in paket namestiti. Enostaven ukaz

```
npm install [imePaketaAliKnjiznice]
```

namesto razvijalca opravi vse potrebno za namestitev in uvoz paketa oziroma knjižnice v projekt.

3.1.1 Večnitnost vs. dogodki

Razvijalci aplikacij, ki se ukvarjajo z več vhodno-izhodnimi viri, kot je na primer večje število strežnikov, ki skrbijo za več povezav odjemalcev naenkrat, so dolgo prisegali na večnitnost. Večnitnost je (bila) priljubljena zato, ker omogoča razdelitev aplikacije na več vzporednih sodelujočih aktivnosti, to pa pomeni enostavnejšo programsko logiko, vzdrževanje in tudi hitrejše izvajanje aplikacije. Večnitnost aplikacijam, kot so spletni strežniki, ki imajo veliko dela z vhodno-izhodnimi operacijami, omogoča učinkovitejšo rabo procesorja. Dandanes je večnitnost za sodobne procesorje nekaj povsem običajnega, saj vsako jedro procesorja izvaja akcije svoje niti. Negativna

stran večnitnosti je precej zahtevnejše programiranje. Posledično se je začelo uveljavljati dogodkovno vodeno programiranje. Dogodki v aplikaciji namreč omogočajo razvijalcem učinkovitejšo alternativo, ki zagotavlja več nadzora nad menjanjem aktivnosti aplikacije. V aplikaciji, ki temelji na dogodkovnem programiranju, se v ozadju izvaja glavna zanka programa, ki posluša, kdaj uporabnik sproži nek dogodek, in izvede akcijo, ki je predvidena ob takem dogodku. Najpogostejši primeri uporabe dogodkovnega programiranja so grafični uporabniški vmesniki, kjer uporabniki pritiskajo na gumbe, izbirna polja ipd. in s tem sprožajo odzive (ang. callbacks) na te dogodke.

Poleg izbire osnovnega načina razvoja omenjene aplikacije je pomembna še izbira programskega jezika. Zaradi lastnosti, naštetih v naslednjem podpoglavju, sem se odločil za programski jezik MeteorJS.

3.2 MeteorJS

MeteorJS oziroma Meteor je brezplačno in odprtokodno JavaScript ogrodje, napisano z Node.js. Uradno je bil prvič predstavljen konec leta 2011 kot „Skybreak“, v začetku leta 2012 pa so ga preimenovali v „Meteor“. V sredini leta 2012 so ga uspešno razvili in predstavili v „Y Combinator“ inkubatorju.

Sledi opis sedmih razlogov, zaradi katerih sem pri razvoju aplikacije izbral ogrodje Meteor in so enaki kot pri drugih avtorjih, ki ga priporočajo ([7]).

3.2.1 Aplikacije, ki delujejo v realnem času

V zadnjih letih so podjetja, kot so Facebook, Twitter ipd., začela razvijati svoja ogrodja, ki stremijo k izvajanju aplikacij in pridobivanju, upravljanju ter uporabljanju podatkov v realnem času. Za ostala ogrodja, med drugimi tudi Meteor, je to pomenilo, da so jih morali razvijalci hitro in učinkovito prilagoditi, če so želeli, da ostanejo med aktualnimi ogrodji in se ne pridružijo precej veliki množici ostalih JavaScript ogrodij, ki vsakih nekaj let kot „novi in boljši“ prodrejo na tržišče, vendar precej takih ogrodij hitro zatone v pozabo.

Vedno več uporabnikov se želi izogniti t.i. osveževanju strani in po nepotrebnem izgubljati čas, kjer to ni potrebno. Meteor ima v svojem jedru mehanizme, ki skrbijo, da se vse dogaja skoraj v realnem času, torej s približno sekundno zakasnitvijo od sprememb v kodi ali bazi podatkov, brez nepotrebnega osveževanja strani. Razvijalcem Meteor aplikacij se s tem ni potrebno obremenjevati, saj aplikacije, napisane v tem ogrodju, temeljijo na modelu „objavi–naroči“ (ang. publish–subscribe model; prevod po [8]), kjer je vsak odjemalec že privzeto povezan preko svojega vtičnika (ang. socket).

„Objavi–naroči“ je model, pri katerem pošiljatelj sporočila (ang. publisher) ne pošlje sporočila določenemu naslovniku (ang. subscriber), temveč sporočilo okategorizira v nek razred brez vedenja, kateri naslovnik bo sporočilo prejel in kateri ne (če sploh kdo), nato pa to sporočilo pošlje strežniku. Na podoben način prejemnik prejme sporočilo, ki je okategorizirano v razred, ki mu ustreza in ga pričakuje, ne da bi vedel, kdo je pošiljatelj.

3.2.2 Samo en razvijalski jezik za vse platforme

Ena najzahtevnejših stvari za razvijalca spletnih aplikacij je znanje različnih programskih jezikov: skrbeti mora za izgled (ang. frontend), zaledje (ang. backend) in bazo podatkov. Pogosto namreč te tri komponente zahtevajo znanje več kot le enega programskega jezika. Razvijalci Meteor ogrodja so to poenostavili do te mere, da lahko razvijalci Meteor aplikacij kodirajo vse tri komponente izključno z znanjem programskega jezika JavaScript.

Kot primer si lahko ogledamo del kode iz aplikacije Algomed – orodja, ki sem ga razvil v okviru diplomskega dela.

```
// collections.js
...
Diseases = new orion.collection('diseases', { /* moznosti, kot so
    naslov, ime v ednini, mnozini itd. */ }); // OrionJS
// Diseases = new Meteor.collection('diseases');  // MeteorJS
...
```

Sam sem sicer zaradi uporabe knjižnice OrionJS uporabil nekoliko drugačno sintakso, se pa v ozadju izvede ukaz, ki je v ogrodju Meteor ekvivalenten kreaciji tabele v jeziku SQL (ustvari se t.i. zbirka). V tem primeru lahko vidimo, da je potrebno le znanje jezika JavaScript in sintakse ogrodja Meteor, ne pa tudi jezika SQL, za upravljanje z bazo podatkov. Na podoben način, vidno na naslednjem primeru, vstavljamo nove ali pa urejamo in brišemo obstoječe podatke.

```
// new-algorithm.js, primer vstavljanja zapisa
...
Notifications.insert({
   text: 'V cakalni vrsti imate nov algoritem',
   type: NotificationTypes.findOne({value: 2})._id,
   seen_ids: {
      ids: [Meteor.userId()]
   }
});
...
```

Poleg možnega urejanja vseh treh omenjenih komponent je prednost v tem, da lahko identična koda teče tako na strežniku kot odjemalcu in dela drugačni stvari.

Za primer vzemimo kreacijo zbirke. Na strežniku se, kot že ukaz pove, ustvari nova zbirka. Na odjemalčevi strani pa se v brskalniku ustvari lokalna zbirka, s katero odjemalec lahko komunicira. Podatki na odjemalčevi strani se zato lahko spremenijo v trenutku, medtem ko se sprememba na strežniški strani zgodi skoraj sinhrono, le z malenkostno zakasnitvijo. Odjemalec posledično vidi in uporablja nove podatke, medtem ko se strežnikovi podatki še sinhronizirajo z odjemalčevimi. Kot že prej omenjeno, se razvijalcu aplikacije s tem ni potrebno obremenjevati, saj za vse poskrbi Meteor sam.

3.2.3 Pametni programski paketi in knjižnice

Meteor vsebuje vgrajene pakete in zunanje knjižnice, ki jih je mogoče vključiti v projekt, kot na primer v programskem jeziku Java.

```
import java.util.Scanner; /* primer uvoza razreda Scanner v .java
    datoteko */
meteor add ejson /* primer uvoza paketa EJSON v celoten projekt.
    Ukaz je zagnan iz ukazne vrstice */
```

Ti paketi omogočajo precej enostavnejšo implementacijo sicer zahtevnih problemov. Vzemimo za primer implementacijo uporabniških računov v aplikaciji. Tega bi se v večini programskih jezikov lotili tako, da bi ustvarili tabelo (v Meteorju zbirko), ki bi vsebovala te podatke, za pozabljena gesla in podobne stvari pa bi potrebovali precej razmišljanja in načrtovanja, da bi dosegli željeni učinek, kot ga vidimo na spletnih aplikacijah, kot so Gmail, Facebook ipd. V Meteorju je to precej enostaven proces, saj enostavno uvozimo pametni paket v projekt. Naslednji ukaz poženemo iz ukazne lupine, ta pa doda množico funkcij (prikazano na primeru A.2), ki so že pripravljene za uporabo, doda pa tudi celoten uporabniški sistem, ki pri vsakem kreiranju novega uporabnika pričakuje e-poštni naslov (ali uporabniško ime) in geslo.

```
// ukazna lupina
meteor add accounts-password
```

Obstajajo tudi paketi, ki omogočajo povezavo s Facebookom (accounts-facebook), Twitterjem (account-twitter), Googlom (accounts-google), pa tudi poseben paket, ki celotnemu uporabniškemu sistemu prilagaja videz aplikacije (accounts-ui). Naslednji primer prikazuje del programske kode, s katero strežnik preveri, če je v sistem prijavljen uporabnik, in naloži ustrezen segment kode.

```
/* currentUser je rezervirana beseda paketa accounts-ui, ki
   preveri, ce je uporabnik vpisan in vrne logicni True ali False
   */
{{#if currentUser}}
  <p>Uporabnik je vpisan</p>
{{else}}
  <p>Uporabnik ni vpisan</p>
{{/if}}
```

Poleg paketov za implementacijo uporabniškega sistema je na voljo več kot 11.000 paketov, kot so npr. Twitter Bootstrap za oblikovanje stilov, Underscore za krajšo kodo in vnaprej napisane funkcije, LESS za naprednejši CSS zapis, EJSON za razčlenjevanje JSON datotek in drugi.

V primeru, da niti med vsemi paketi ne najdemo tistega, ki nam ustreza, je tu še nepregledno število knjižnic, ki jih vsebuje NPM in jih brez težav dodamo ter uporabljamo v projektu.

3.2.4 Logični operatorji za prikaz elementov in t.i. helperji

Meteor omogoča poleg pisanja logičnih operatorjev v JavaScript datotekah tudi pisanje logičnih operatorjev v HTML datotekah.

Primer A.3 prikazuje del kode za prikaz gumba „Čakalna vrsta“ orodne vrstice. Iz primera je razvidno, da je gumb viden samo v primeru, če je uporabniku dodeljena ustrezna uporabniška vloga, katere namen je sprejemanje in zavračanje spremenjenih ali novih algoritmov. Omenjeni primer prikazuje tudi uporabo Meteorjevih t.i. „helperjev“.

Helperji (JavaScript) predstavljajo objekt predloge, čigar ključi so raznorazne funkcije. Helper (HTML) je ime zaledne funkcije, ki je hkrati povezava med HTML in JavaScript kodo bodisi za dinamičen prikaz podatkov v HTML značkah bodisi prikazovanje DOM elementov glede na izid JavaScript funkcij v zaledju.

Na primeru A.3 je viden helper `roleIsCommision`, kar je ime funkcije v zaledju, medtem ko primer A.4 prikazuje zaledni del uporabe helperja `roleIsCommision`.

Funkcija `roleIsCommision` vrne rezultat `True` v primeru, da je trenutni prijavljeni uporabnik član komisije, ki skrbi za sprejemanje ali zavračanje novih algoritmov (v podatkovni bazi so člani komisije označeni z vrednostjo 2), sicer vrne rezultat `False`.

3.2.5 Dobro podprti forum, uporabniška skupnost in dokumentacija

Ob izbiri programskega jezika in ogrodja pogosto k odločitvi prispeva podrobna in razumljiva dokumentacija ter podpora in pomoč uporabnikom. Razvijalci Meteorja so zagotovili podrobno dokumentacijo podkrepljeno s primeri uporabe. Skupina Crater obvešča o programerskih novicah, na spletu pa je ogromno posnetkov, ki prikazujejo implementacijo tako enostavnih kot težjih rešitev programerskih problemov. Tudi sam razvoj Meteorja gre v pravo smer, saj naraščajo tako število uporabnikov kot tudi ekipe za razvoj ogrodja in pomoč uporabnikom.

3.2.6 Razvoj v skladu z razvijalskimi željami

Praktično vsi programski jeziki in ogrodja v ospredje postavljajo svoje prednosti, češ da so enostavnejši in učinkovitejši kot ostali, pri nemalem številu od njih pa opazimo, da je precej drugače. Razvijalci Meteorja so razvili sistem, ki omogoča enostavno programiranje razvijalcem vseh stopenj.

Za primer vzemimo del kode:

```
// home.html
<template name="home_index">
  {{> navbar}}
  {{> home}}
</template>

<template name="home">
   <div id="main-container" class="container">
      <div class="well" id="home-well">
         <h2>Dobrodosli, {{userFirstName}}</h2>
      </div>
   </div>
</div>
```

V tem primeru lahko vidimo, da kot razvijalci ne potrebujemo vključevanja HTML značk, prav tako ne potrebujemo vključevanja CSS in JavaScript datotek – za vse poskrbi Meteor sam. Seveda to ni nekaj, kar bi ločevalo dobra ogrodja od slabih, je pa vsekakor dobrodošlo.

Še nekatere glavne prednosti so:

1. Samodejno osveževanje strani v brskalniku ob shranjevanju sprememb v kodi.

2. Prosta pot o odločanju glede strukture projekta – obstajajo standardi in predlogi, vendar ne zahtevajo, da se jih kot razvijalci držimo.

3. Zgradba spletne strani temelji na osnovi predlog (ang. template). V zadnjem primeru vidimo, da s preprostim ukazom

   ```
   {{> navbar}}
   ```

   orodno vrstico vključimo na vrh strani, ne da bi še enkrat pisali celotno kodo in stile tam, kjer potrebujemo orodno vrstico. Predlogo z imenom

„navbar“ smo seveda prej definirali, ji določili stile in odzive na gumbe ter druge grafične komponente.

3.2.7 Preprosto ogrodje tudi za začetnike

Zaradi svoje MVC (ang. Model-View-Controller) strukture je Meteor dobro orodje tudi za začetnike, saj se na ta način izogne mešanju kode za izgled, stile in zaledje aplikacije.

MVC označuje strukturni model aplikacije, ki aplikacijo logično razdeli na tri dele: model, pogled in krmilnik. Model je zadolžen za podatkovno bazo in logiko aplikacije, pogled skrbi za vizualno predstavitev podatkov in informacij (kar na koncu uporabniki vidijo na zaslonu), krmilnik pa sprejme ukaz in ustrezno upravlja bodisi s pogledom bodisi z modelom.

Primer A.5 prikazuje implementacijo gumba v ospredju aplikacije (View) in njegov odziv na pritisk v zaledju aplikacije (Controller).

Poglavje 4

Razvoj in implementacija orodja Algomed

V času študija sem spoznal več načinov razvoja programske opreme, med katerimi so me najbolj privlačile agilne metodologije – natančneje: agilna metoda Scrum ([9]). Prav tako sem kot sodelavec pri poslovnih projektih ves čas deležen razvoja z omenjeno metodo, zato sem se odločil, da bom preizkušeni metodi sledil tudi v razvoju Algomeda.

4.1 Razvoj aplikacije Algomed

Med razvojem aplikacije Algomed sem sledil fazam razvoja programske opreme, definirane s standardom ANSI/IEEE 792-1983. Celotni cikel razvoja programske opreme vključuje naslednje osnovne faze:

- analiza zahtev
- načrtovanje
- implementacija
- testiranje
- vzdrževanje

Standard vsebuje tudi fazi „namestitev“ (sledi testiranju) in „umik iz uporabe“ (sledi vzdrževanju), vendar bom povzel le prve tri faze in povzetke podprl s primeri iz razvite aplikacije.

Analiza zahtev

Analiza zahtev je faza, v kateri od naročnika prejmemo zahteve o izgledu, funkcionalnosti in ostalih specifikacijah, ki naj bi jih zagotavljala naročena programska oprema. Če naročnika ni, ima razvijalec prosto pot in si zahteve postavlja sam. Glavno vprašanje, ki si ga moramo med analizo zahtev zadati, je, kaj naj bi programska oprema zagotavljala. Na podlagi pričakovanj, ki jih je izrazil urednik slovenskih medicinskih algoritmov ([3], [4]) sem oblikoval osnovne zahteve, ki jih prikazuje tabela 4.1.

#	Naslov uporabniške zgodbe	Podrobnosti
1	Digitalizacija algoritmov	Preslikava diagnostičnih in terapevtskih algoritmov iz tiskane v elektronsko obliko
2	Ustvarjanje algoritmov	Možnost dodajanja novih algoritmov
3	Urejanje algoritmov	Možnost urejanja vsakega polja algoritma in dodajanja novih polj v potek algoritma
4	Hiter dostop do podrobnosti vsakega polja	Z dotikom ali klikom (odvisno od naprave) na določeno polje algoritma naj se prikažejo podrobnosti tega polja
5	Zunanji viri	Sistem naj vsebuje povezave do zunanjih spletnih podatkovnih baz (npr. zdravil)
6	Aplikacija neodvisna od sistema	Aplikacija mora biti dostopna preko računalnika, tablice in pametnega mobilnega telefona, operacijski sistem pri tem ne sme igrati vloge
7	Uporabniku prijazen sistem	Izgled in interaktivnost morata biti uporabniku prijazna in razumljiva
8	Sistem za sprejemanje in zavračanje algoritmov	Obstajati mora sistem (organ), ki bo zagotavljal, da so novi ali spremenjeni obstoječi algoritmi strokovno pregledani in nato bodisi sprejeti bodisi zavrnjeni

Tabela 4.1: Tabela uporabniških zgodb zajetih med analizo zahtev

Načrtovanje

Vrsticam tabele, ki sem jih pridobil z analizo zahtev, sem v nadaljevanju pripisal stopnjo pomembnosti, načrtovani čas implementacije in kratek opis implementacije. Tako sem razvrstil izzive od najzahtevnejših do najenostavnejših. Po končani fazi načrtovanja sem vse tabele prepisal v sistem za sledenje uporabniškim zgodbam spletnega sistema BitBucket.

Poleg izbire orodja (Node.js in MeteorJS) je bilo pomembno vprašanje, katero podatkovno bazo izbrati za razvoj. Zaradi najpreprostejše tekstovne predstavitve algoritmov bolezni in terapij sem izbral MongoDB.

4.2 Implementacija aplikacije Algomed

Faza implementacije zajema kodiranje. Kodiranje pomeni prepis zastavljenih algoritmov v programsko kodo z uporabo sintakse izbranega programskega jezika.

4.2.1 Agilni pristop razvoja programske opreme

Orodja sem se lotil z uporabo nekaterih principov agilnih metodologij.

Inkrementalni razvoj

Že pred začetkom izdelave projekta sem ustvaril nov Git repozitorij, namenjen shranjevanju programske kode na oddaljen strežnik. Celotno aplikacijo sem razvijal v številnih kratkih ciklih, vsak cikel pa je na koncu vseboval delujočo in uporabno kodo. Posledično je projekt razpadel v zaporedje obvladljivih delov. Dodatna prednost je bila tudi možnost nadaljevanja projekta na drugih uporabniških zgodbah, če je bila katera prezahtevna za kratkotrajen razvoj.

Enostavnost

Funkcije, ki sem jih pisal, so dokaj preproste in njihovo spreminjanje ni težavno.

Prilagodljivost

Celotna aplikacija je zgrajena tako, da omogoča nezahtevno implementacijo nadgradenj, kjerkoli si zaželimo.

4.2.2 Osnovna spletna stran

Razvoja sem se lotil s postavitvijo osnovne spletne strani. Ker je bila uporabniška zgodba, ustvarjena za določanje izgleda celotne aplikacije, na seznamu s prioritetami dokaj nizko, sem celoten izgled kar dolgo ohranil v t.i. žičnem modelu brez določenih barv in z obrobami na vseh komponentah.

4.2.3 Uporabniški sistem in podatkovna baza

Po kreaciji osnovne spletne strani sem se lotil implementacije uporabniškega sistema. Kot že omenjeno, vsebuje Meteor pametne pakete, ki skrbijo za enostavnejši razvoj tega sicer zahtevnega problema.

Z vključitvijo paketov „accounts-password“ in „accounts-ui“ je bil celoten uporabniški sistem končan. Ker sem aplikacijo zastavil tako, da od uporabnika potrebujem več podrobnosti kot le e-poštni naslov (ali uporabniško ime) in geslo, sem izdelal tudi podatkovno bazo.

OrionJS

Že pred razvojem Algomeda sem tako za osebni kot tudi poslovni projekt uporabljal knjižnico OrionJS (krajše Orion). Orion je odprtokodna, dokaj preprosta, a izjemno močna knjižnica za integracijo administratorske strani in nekaterih osnovnih funkcionalnosti za izgradnjo spletne strani na enostavnejši način. Poleg administratorske strani in osnov Orion skrbi za enostavnejši in

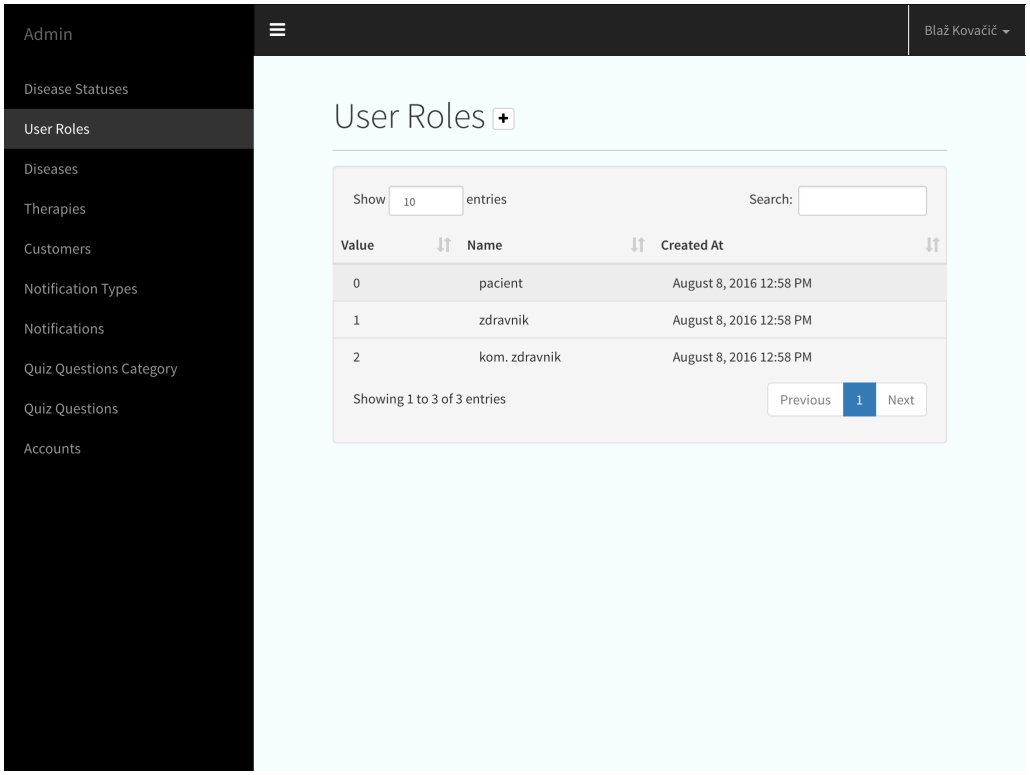
lažje predstavljiv sistem za implementacijo podatkovne baze. Ker temelji na Meteorju, je namenjen tudi za razvoj programske opreme, ki deluje na vseh večjih platformah.

Zaradi svojih temeljev se tudi Orion na nek način obnaša kot samostojno ogrodje. Meteor ima možnost uvoza raznoraznih pametnih paketov in knjižnic, kar pomeni, da ima Orion zajetno množico pametnih paketov za uvoz v projekt.

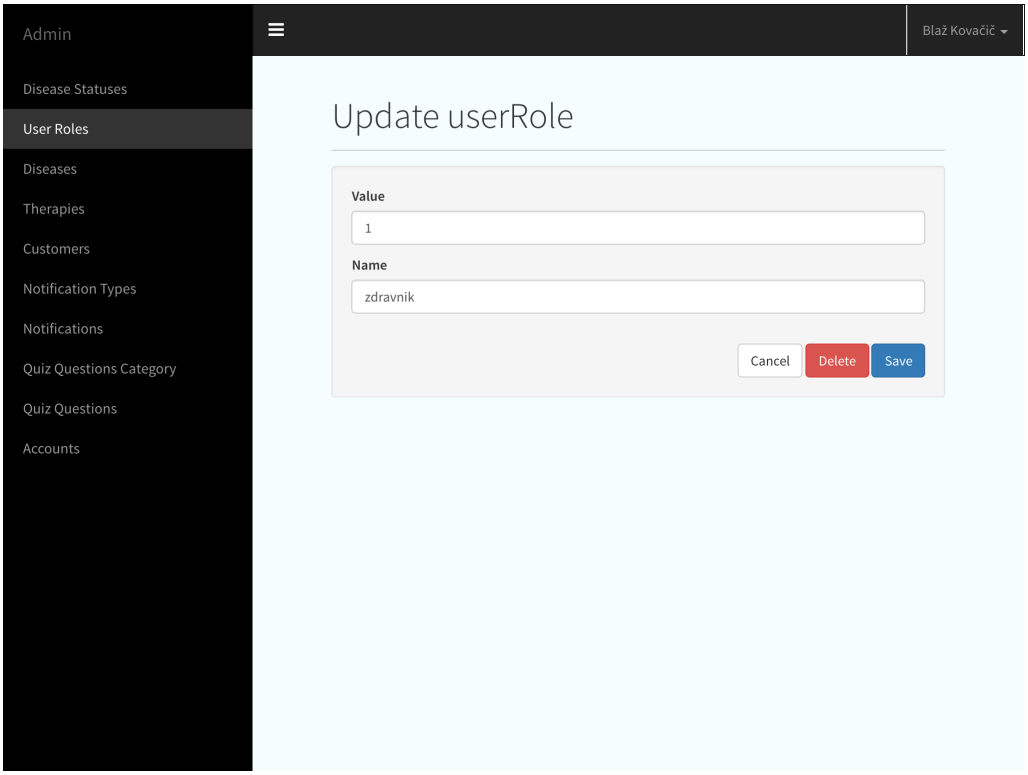
Z vključitvijo OrionJS knjižnice sem se lotil kreacije vseh potrebnih zbirk za začetno bazo podatkov. Ustvaril sem zbirko „Customers“, saj je zbirka „Users“ rezervirana zbirka za Meteorjev uporabniški sistem. V zbirko Customers sem nato preko polja „user_id“ povezal račun, ki ga ustvari Meteor sam. Na tak način sem dobil dve zbirki, namenjeni uporabnikom. Ena vsebuje objekte, ki vsebujejo enolični identifikator, e-poštni naslov (ali uporabniško ime) in geslo – po želji tudi polje „profile“, z raznoraznimi podatki. Druga vsebuje objekte z enoličnimi identifikatorji, raznimi podatki, kot so ime, priimek, varnostna vprašanja in odgovori (v uporabi za pozabljeno geslo), na koncu pa, kot že omenjeno, „user_id“, ki vsebuje identifikator za istega uporabnika v zbirki „Users“. Primer kreacije zbirke Customers prikazuje primer A.1.

Sistem Algomed temelji tudi na različnih vlogah, ki jih imajo uporabniki. Pacientom in/ali študentom sistem prikaže algoritme brez možnosti urejanja in dodajanja novih, lahko pa rešijo kviz, ki služi za utrjevanje znanja. Zdravnikom se poleg omenjenih možnosti prikaže tudi možnost dodajanja novih algoritmov; zdravnikom, ki so člani komisije za sprejemanje in zavračanje novih ali spremenjenih algoritmov, pa se dodatno prikaže tudi čakalna vrsta tovrstnih algoritmov, ki čakajo na odločitev. Da sem lahko to zagotovil, sem dodal tudi novo zbirko „User Roles“, nato pa preko polja „role“ objekta zbirke Customers z razmerjem „has one“ povezal ustrezno vlogo. Za dodajanje relacij v NoSQL bazi podatkov sem moral predhodno dodati pametni paket „Orion Relationships“. NoSQL označuje sistem za bazo podatkov, ki ne temelji na relacijskem modelu.

Naslednji sliki prikazujeta administratorski sistem, ki ga Orion samodejno zgradi in z ustvarjanjem zbirk tudi dopolnjuje.



Slika 4.1: Administratorski sistem knjižnice OrionJS. Na sliki je prikazan pregled zbirke uporabniških vlog. Vsako vlogo je mogoče urejati in brisati (polja so določena pri kreaciji zbirke), mogoče pa je dodati tudi novo vlogo. Prikaz teh možnosti prikazuje slika 4.2.



Slika 4.2: Pritisk na določeno vlogo prikaže podrobnosti, ki se jih da urejati in akcijske gumbe za preklic urejanja, brisanje podatka ali shrambo sprememb.

Sprejemanje in zavračanje novih ali spremenjenih obstoječih algoritmov

Uporabniško zgodbo, v tabeli 4.1 označeni s številko 8, sem implementiral z uporabo zbirke uporabniških vlog.

Razvoj podatkovne baze sem nadaljeval s kreacijo zbirk „Algorithm Statuses“, „Diseases“ in „Therapies“. Zbirka „Algorithm Statuses“ vsebuje možna stanja (statuse), ki jih zajemajo algoritmi.

V zbirko sem dodal tri vloge in sicer: pacient, zdravnik in član komisije. Slednji skrbi za sprejemanje ali zavračanje novih in spremenjenih obstoječih algoritmov.

V trenutku, ko zdravnik ali član komisije predlaga spremembo algoritma

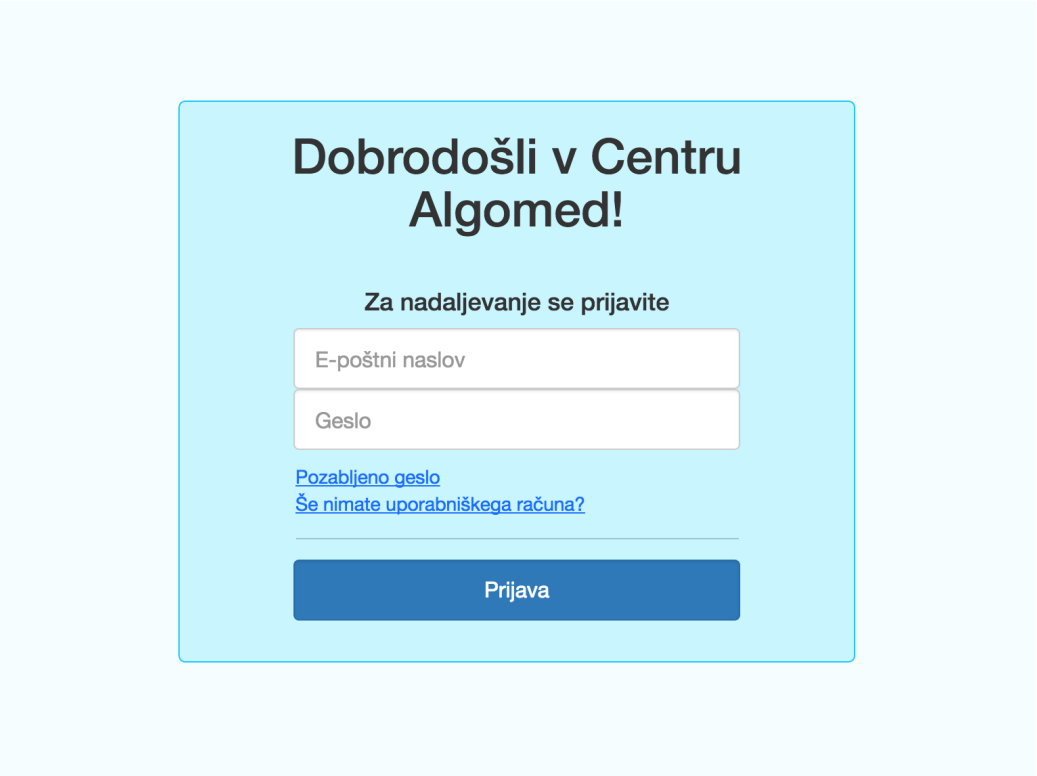
in ga pošlje v pregled za odobritev, se ostalim članom komisije spremenjeni algoritem prikaže v čakalni vrsti algoritmov. Možnost pregleda teh predlogov imajo samo člani komisije. Medtem, ko je v čakalni vrsti shranjen spremenjen algoritem, se v primeru, da gre za spremenjen že obstoječi in ne nov algoritem, na naslovni poti `/algorithmEditor/[identifikatorTegaAlgoritma]` še vedno prikaže sprejeta verzija algoritma, da lahko z njim operirajo vsi ostali uporabniki. Izida za algoritme v čakalni vrsti sta dva, izid pa določi večina komisije. Algoritem lahko pridobi status „sprejet“, spremenjeni algoritem pa gre neposredno v uporabo. Če je predlog s strani večine komisije zavrnjen pa se v primeru, da gre za spremenjen obstoječi algoritem, še naprej uporablja obstoječa verzija, zavrnjeni predlog novega algoritma pa obtiči v bazi podatkov.

Therapies vsebuje algoritme za zdravljenje notranjih bolezni, opisane v prvem priročniku ([3]), Diseases pa algoritme za razpoznavanje notranjih bolezni v drugem priročniku ([4]).

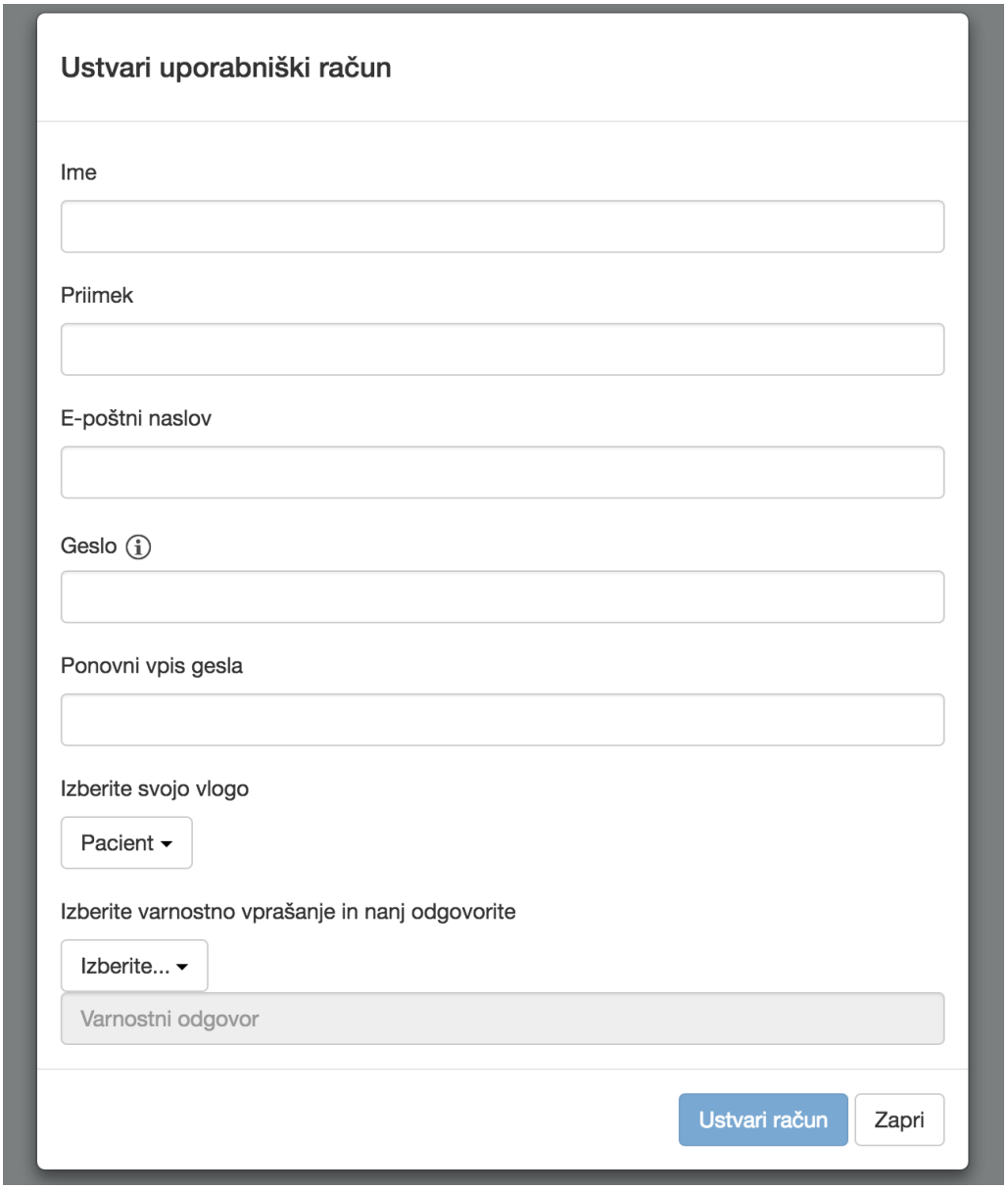
Na koncu sem dodal še zbirko „NotificationTypes“ in „Notifications“. Notifications vsebuje obvestila, ki jih prejemajo uporabniki, vsak objekt te zbirke pa je preko polja „type“ z relacijo „has-one“ povezan z zbirko NotificationTypes. NotificationTypes vsebujejo možne tipe za polje „type“ objekta zbirke Notifications, ki tudi določajo izgled obvestila.

Po razvoju podatkovne baze sem aplikaciji dodal sistem za registracijo in prijavo uporabnikov. V obeh primerih sem uporabil modalna pojavitvena okna, ki jih omogoča pametni paket „Bootstrap 3 Modal“. Njihov namen je jasen in enostaven prikaz raznoraznih form, podatkov, tabel, slik in ostalega. V mojem primeru modalna pojavitvena okna služijo formam, ki so namenjene registraciji uporabnika, obnovitvi pozabljenega gesla in spremembi gesla. Naslednji sklop slik prikazuje segment aplikacije, namenjen prijavi v sistem in uporabi modalnih pojavitvenih oken pri registraciji uporabnika ali v primeru težav ob prijavi v sistem.

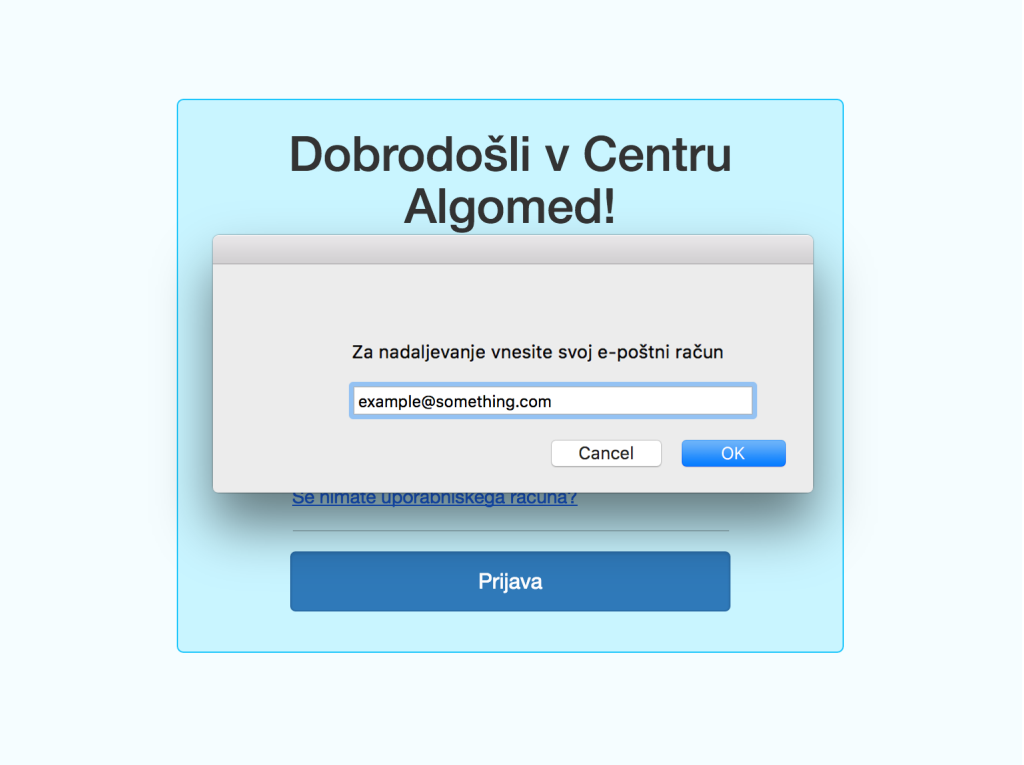
Celotna aplikacija je zasnovana, da se izgled prilagaja velikosti zaslona, tako da deluje na vseh računalnikih in mobilnih napravah.



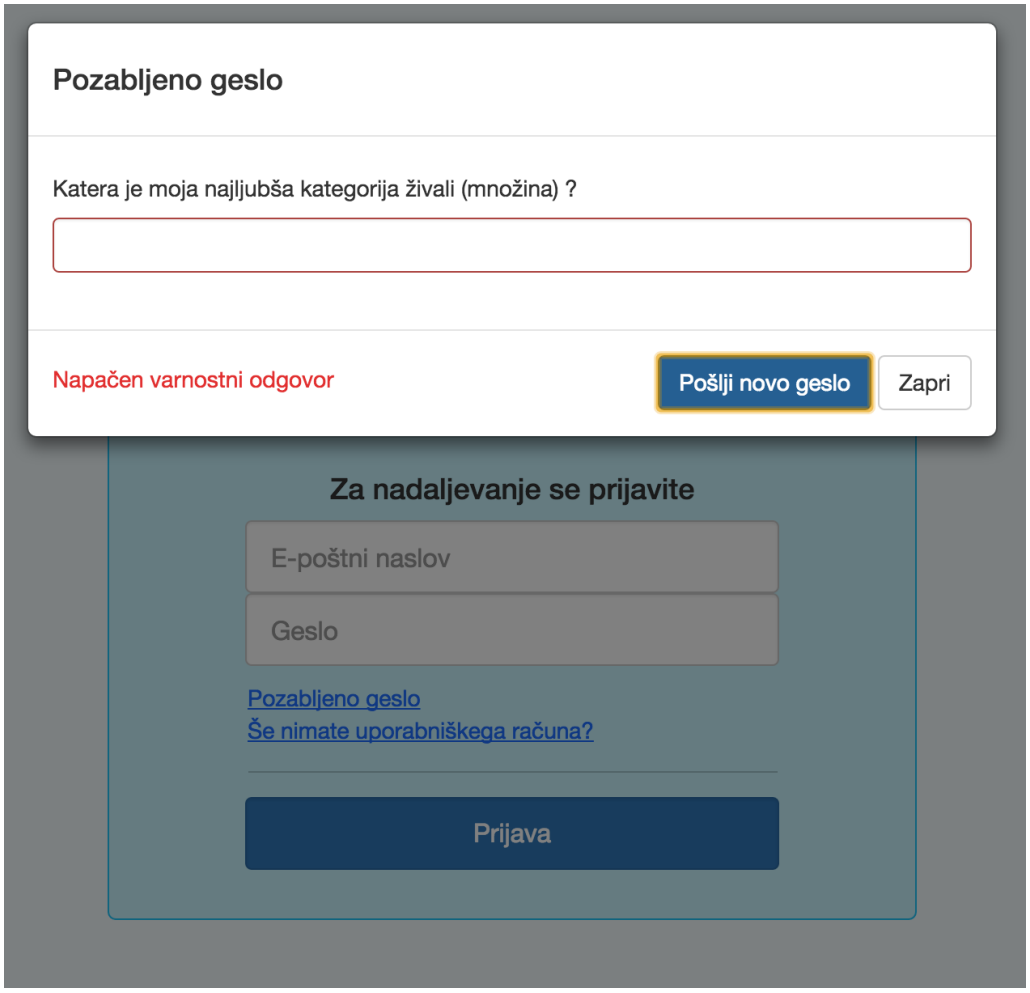
Slika 4.3: Forma za prijavo v sistem in izbiro ostalih možnosti glede uporabniškega računa. Ob kliku na možnost „Prijava“ se na odjemalčevi strani izvede funkcija `Meteor.loginWithPassword(email, password, callback)`, ki preveri obe polji in v primeru uspeha uporabnika vpiše v sistem, v primeru neuspeha pa ga o tem obvesti.



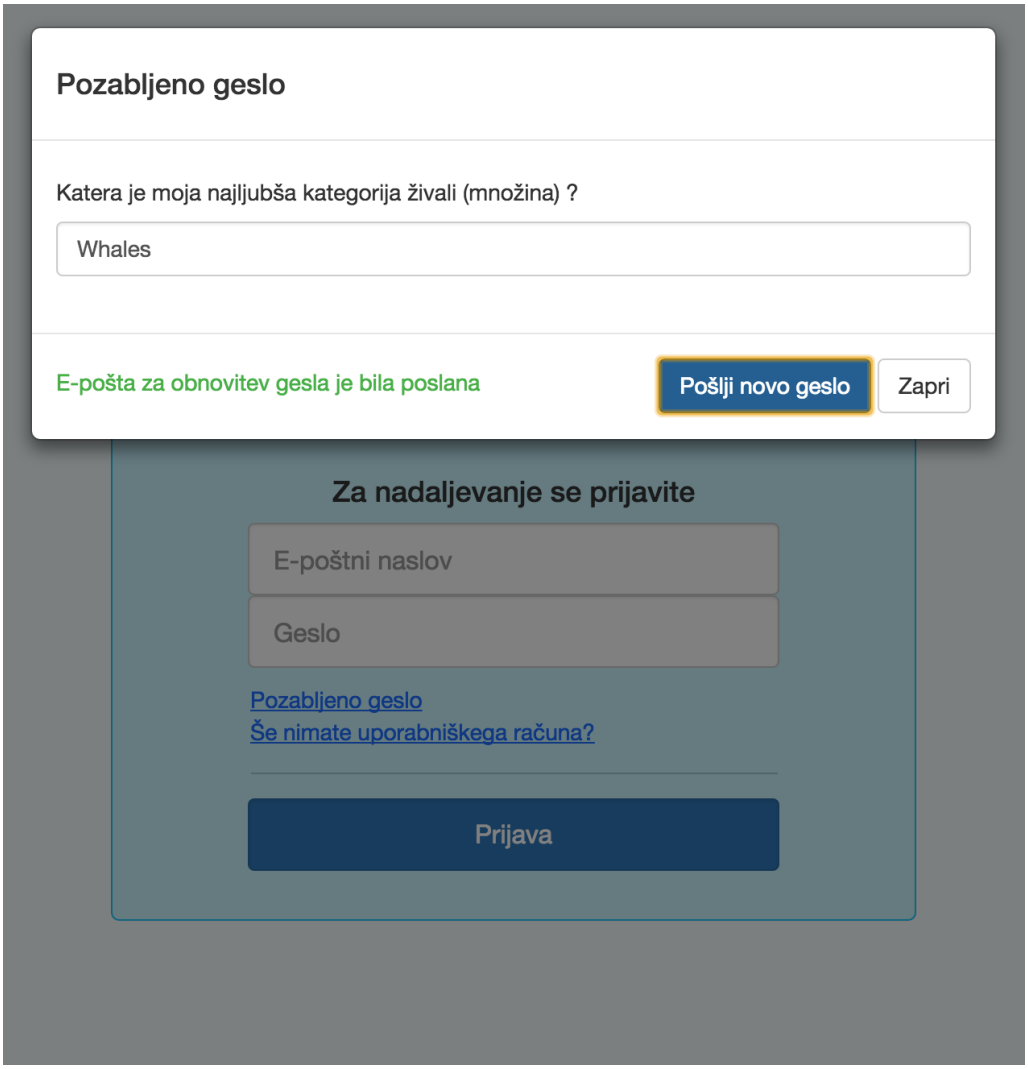
Slika 4.4: Slika prikazuje modalno pojavitveno okno za registracijo uporabnika v sistem Algomed. Uporabnik mora, če želi, da gumb „Ustvari račun“ postane odziven, izpolniti vsa polja. Polja za ime, priimek e-poštni naslov, geslo in ponovni vpis gesla se validirajo. V primeru pravilnih vnosov se ob pritisku na gumb za potrditev registracije izvede registracija uporabnika v sistem, v primeru napake pa se uporabnika obvesti, v katerih poljih se nahaja napaka in uporabnik ni registriran.



Slika 4.5: Privzeto pojavitveno okno v brskalniku Google Chrome. S tem oknom uporabnik vnese svoj e-poštni naslov, s katerim je registriran v sistemu Algomed, da se mu odpre možnost za pošiljanje povezave, namenjene obnovitvi gesla. V primeru vnosa e-poštnega naslova, ki v sistemu še ne obstaja, se uporabnika preko enakega pojavitvenega okna o tem obvesti, možnost obnovitve gesla pa je prekinjena.



Slika 4.6: Modalno pojavitveno okno za uporabnikov vnos varnostnega odgovora na varnostno vprašanje. Tako vprašanje kot odgovor je uporabnik določil med registracijo v sistem, kar je razvidno na sliki 4.4. Na sliki je prikazano stanje, ko uporabnik vnese napačen odgovor in pritisne na potrditveni gumb.



Slika 4.7: Enako pojavitveno okno, kot na sliki 4.6, vendar ta slika prikazuje stanje po pravilnem odgovoru. Z manjhno zakasnitvijo se pojavitveno okno počasi zapre.

Po končani prijavi v sistem sem implementiral profilno stran, kjer lahko uporabniki urejajo svoje osebne podatke, spremenijo varnostno vprašanje in odgovor ter spremenijo geslo za vstop v sistem.

Migracije

Nekatere podatke, kot so uporabniki, vloge uporabnikov, tipi obvestil in nekateri algoritmi za razpoznavanje in zdravljenje, sem definiral že med razvojem aplikacije in jih migriral v bazo podatkov.

Migracija je prenos zapisanih podatkov v dejanske podatke v podatkovni bazi. Sistem migriranja je v Meteorju zelo koristen, saj lahko določimo več verzij migracij in nato s funkcijo

```
Meteor.startup(function() {
  Migrations.migrateTo(X); /* X oznacuje verzijo, lahko je namesto
     X tudi 'latest' */
});
```

ob zagonu strežnika naročimo sistemu, naj podatke, označene z verzijo X (ali „latest“, kar v angleščini pomeni zadnja možna verzija), migrira v bazo podatkov. Tako se lahko ob napaki ali zaradi kakršnega koli drugega razloga kadarkoli vrnemo na željeno verzijo podatkovne baze.

Primer: Implementiramo podatke in verzijo migracije označimo z 1. Primer implementacije migracije je prikazan na primeru A.6. Nato v poteku aplikacije dodamo nove podatke in jih označimo z 2. Kasneje ugotovimo, da smo v zadnji verziji naredili napako in želimo obnoviti stare podatke. V anonimni funkciji, ki jo podamo kot argument funkciji „Meteor.startup“, enostavno izvedemo klic

```
Migrations.migrateTo(1);
```

in v bazo podatkov ponovno dobimo podatke, označene z 1.

4.2.4 Prikaz, urejanje in ustvarjanje algoritmov

Glavni del aplikacije je namenjen prikazu, urejanju in dodajanju novih algoritmov, kar v tabeli 4.1 opisujejo uporabniške zgodbe 1 – 4.

Vprašanje, ki sem si ga zastavil pri načrtovanju najpomembnejšega dela

sistema Algomed, je bilo, kako naj v podatkovni bazi predstavim posamezen algoritem. Ker OrionJS poleg osnovnih podatkovih tipov, kot so String, Number in Boolean, podpira tudi podatkovni tip „Object“ (slo. objekt), sem potreboval samo še mehanizem, ki bi pretvoril objektni zapis algoritma v dejanski objekt, ki bi se shranil v zbirko. Ta mehanizem sem pridobil s pametnim paketom EJSON (Extended JSON). Z njegovo pomočjo sem algoritme, predstavljene v JSON formatu, pretvoril v dejanske objekte in jih shranil v ustrezno zbirko bodisi med diagnoze bodisi med terapije.


```
{
  "name": "Utrujenost",
  "details": "Utrujenost je <b>subjektivni občutek zmanjšane zmogljivosti za telesno
  "children": {
    "type": "choice",
    "text": "Anamneza in fizikalni pregled pozitivna",
    "details": "<b>ANAMNEZA IN KLINIČNI PREGLED:</b> Ker je utrujenost v večini prime
    "values": {
      "DA": {
        "type": "action",
        "text": "Srčno popuščanje<br>Respiratorna insuficienca<br>Zastrupitve<br>Vpli
        "details": "<b>VPLIV ZDRAVIL:</b> Zdravila lahko povzročijo utrujenost z nepo
      },
      "NE": {
        "type": "choice",
        "text": "Osnovne laboratorijske preiskave",
        "details": "<b>LABORATORIJSKE PREISKAVE:</b> Kadar utrujenosti ne spremljajo
        "values": {
          "DA": {
            "type": "action",
            "text": "Anemija<br>Ledvična odpoved<br>Sladkorna bolezen<br>Insuficienca
          },
          "NE": {
            "type": "choice",
            "text": "Testi delovanja ščitnice patološki",
            "values": {
              "DA": {
                "type": "action",
                "text": "Hipotiroza<br>Hipertiroza"
              },
              "NE": {
                "type": "choice",
                "text": "Še vedno sum na organsko bolezen",
                "values": {
```


Slika 4.8: Izrez zapisa algoritma za razpoznavanje bolezni, ki se kaže z utrujenostjo kot vodilnim simptomom v formatu JSON.

Pred prikazom algoritmov sem ustvaril še štiri predloge za polja: začetno polje, izbirno polje, akcijsko polje in končno polje. Dejanskega prikaza algoritmov sem se lotil z uporabo rekurzije. Na začetku sem ustvaril začetno polje z uporabo ukaza:

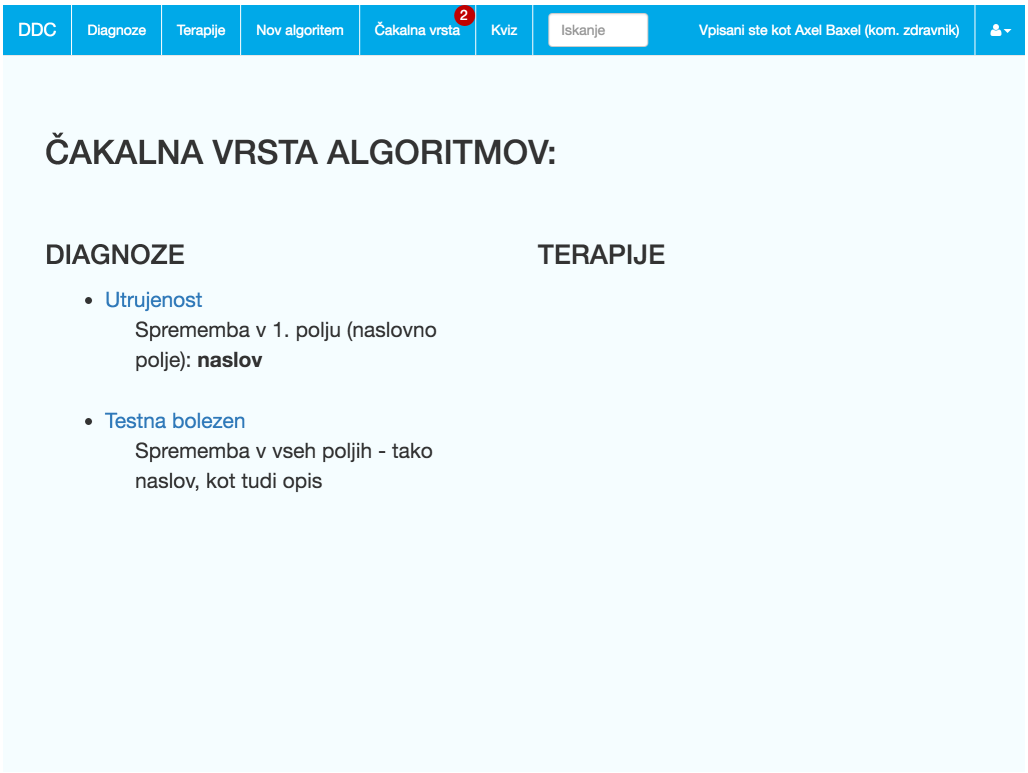
```
// Blaze predstavlja Meteorjev sistem za prikaz elementov
Blaze.renderWithData(
   Template.title_field,  // Predloga za prikaz
   {
      id: "title_field",  // identifikator za polje
      text: html2text(diseaseOrTherapy.name) /* prikazno besedilo v
          polju */
   },
   $(".book-algorithm")[0] /* jQuery selektor za element, kateremu
       naj se kot otrok doda zacetno polje */
);
```

Zatem sem z uporabo rekurzije nadaljeval po objektnem ključu „children“. Akcijsko polje je zavzelo svojo predlogo, izbirno polje je rekurzijo nadaljevalo v kraku vrednosti DA in vrednosti NE, končni polji pa sta zaključili dodajanje polj.

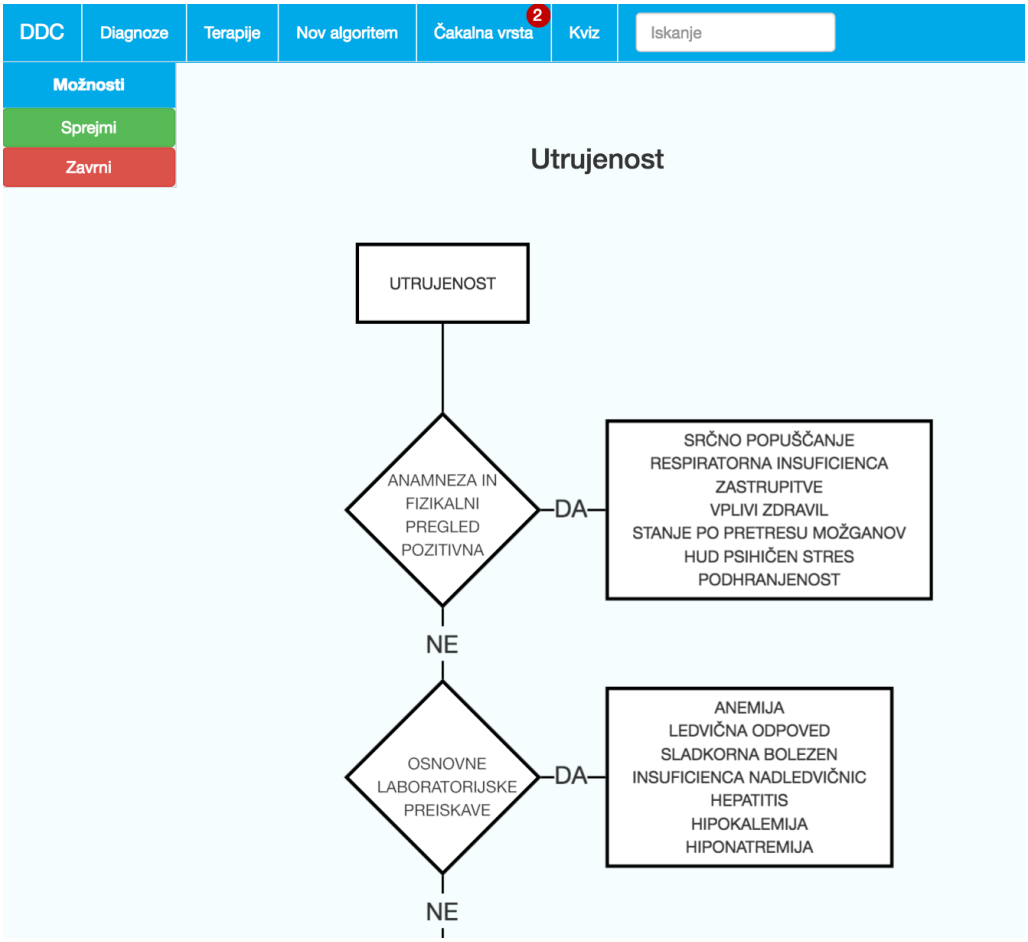
Težave so povzročale povezave med vozlišči. Na začetku sem poskušal z zunanjo knjižnico PlumbJS, namenjeno enostavnemu izrisovanje grafov, nato pa sem zaradi raznoraznih težav raje uporabil HTML platno in sam izrisoval črte. Odločitvena besedila na povezavah so nič drugega kot spustni izbirni meni z možnostima „DA“ in „NE“.

Ker je v aplikaciji del, kjer komisija zavrača in sprejema spremembe o algoritmih, sem prikaz algoritmov, tako obstoječih, kot tudi tistih v čakalni vrsti, moral poenostaviti. Za obe vrsti sem uporabil naslov `/algorithmEditor/[idAlgoritma]`, razlikuje pa se način dostopa do tega naslova. Če je uporabnik do tja prispel preko strani s čakalno vrsto, se namesto orodij za urejanje algoritmov v stranskem meniju pojavita možnosti „Sprejmi“ in „Zavrni“, kot je to prikazano na sliki 4.10. V primeru, da uporabnik kakorkoli drugače

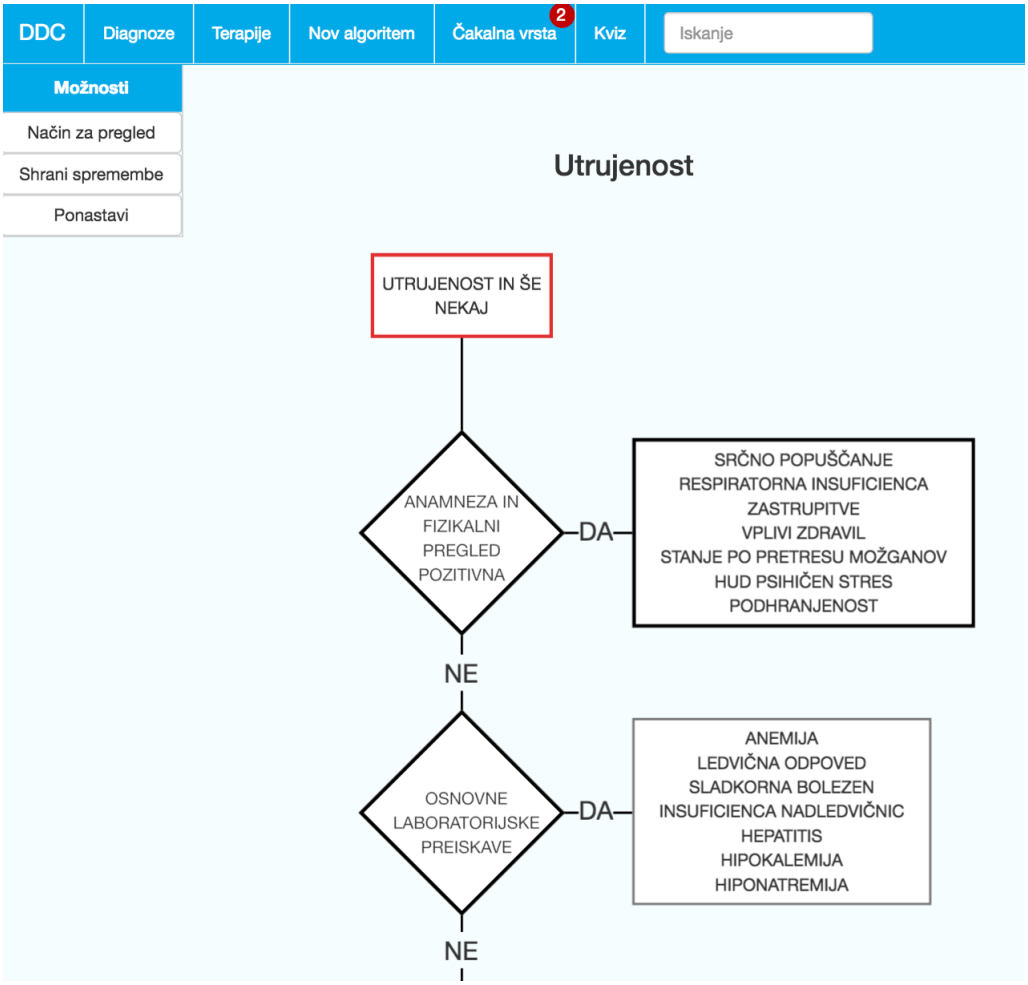
dostopi do istega naslova, pa se mu pokaže algoritem z možnostjo urejanja, vidno na sliki 4.11.



Slika 4.9: Čakalna vrsta spremenjenih ali novih algoritmov.



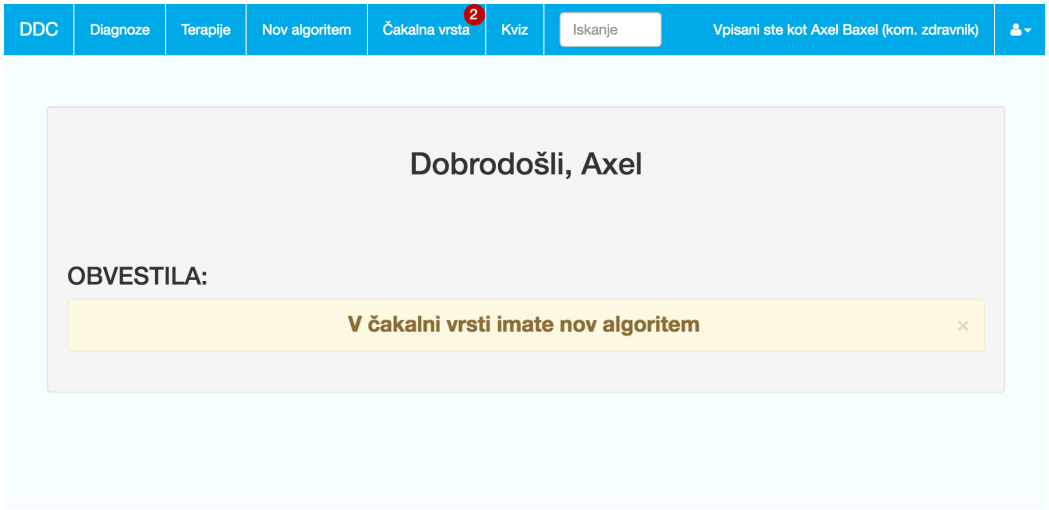
Slika 4.10: Pogled za sprejemanje in zavračanje algoritmov.



Slika 4.11: Pogled za urejanje algoritmov. Algoritem na sliki je imel v začetnem polju besedilo „UTRUJENOST“, po spremembi pa se polje obarva rdeče in sprememba je vidna v trenutku. V primeru pritiska na gumb „Shrani spremembe“, se algoritem prenese v čakalno vrsto, objektu v bazi podatkov pa se doda polje „changes“. Te spremembe so potem vidne v čakalni vrsti, kot je prikazano na sliki 4.9.

V določenih primerih, kot so: registracija v sistem, sprejetje novega algoritma, zavrnitev sprememb itd., se uporabniku na začetni strani prikažejo obvestila (slika 4.12). Dokler uporabnik obvestila ne zapre in s tem potrdi, da ga je prebral, se mu prikaže vedno ob prihodu na domačo stran. Po

pritisku na gumb X se v podatkovno bazo shrani podatek, da je trenutni uporabnik to obvestilo že videl, in obvestilo se mu ne prikaže več.

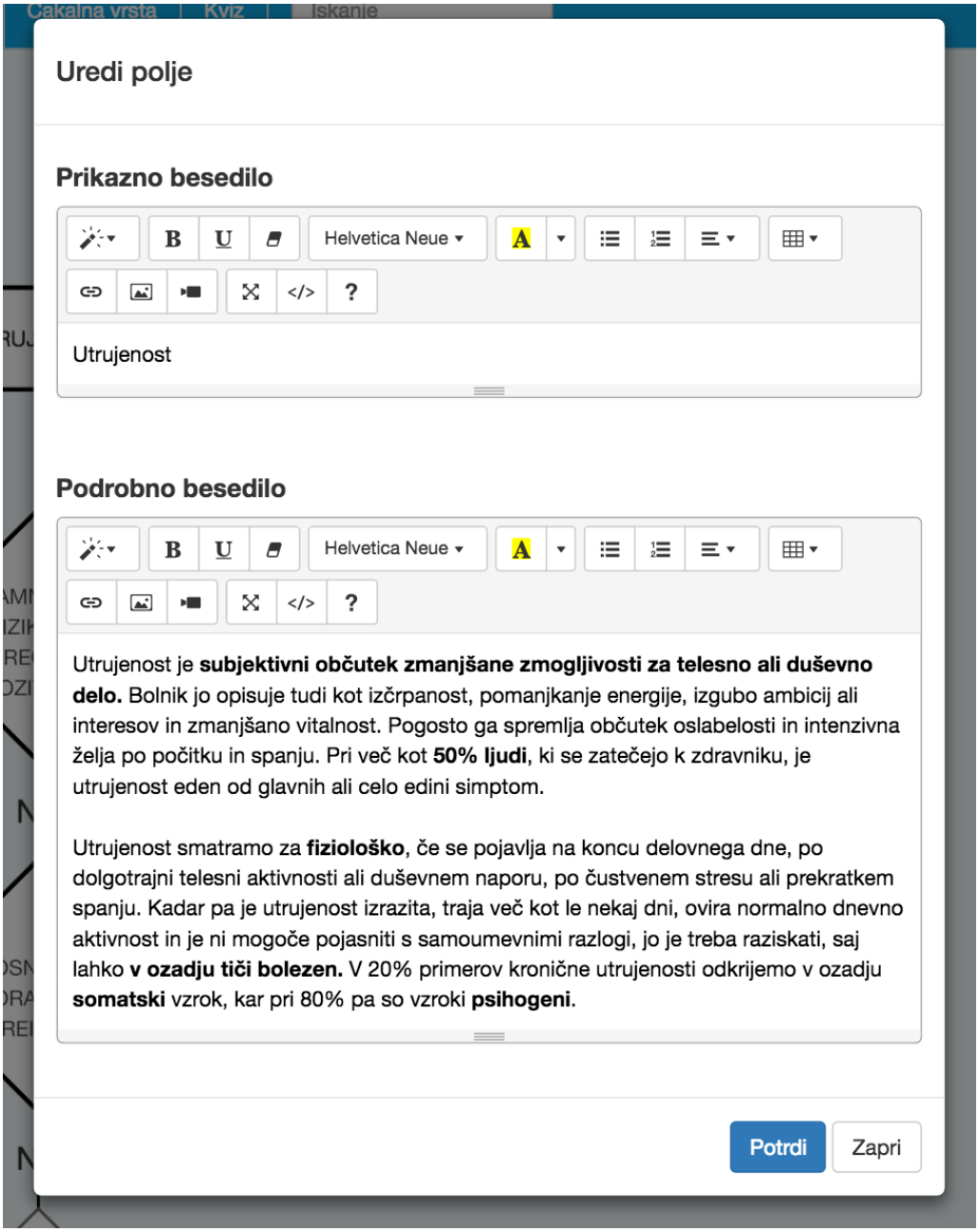


Slika 4.12: Obvestilo o spremembi algoritma Utrujenost.

Urejanje polj

Pri programski opremi je vedno pomembna uporabniška izkušnja, zato sem tudi sam moral poskrbeti za atraktiven izgled. Vprašanje, ki sem si ga zastavil je bilo, kako poskrbeti za dober prikaz polj. V priročnikih so nekatere besede odebeljene, nekatere so napisane z velikimi črkami, ali pa so podčrtane itd. Izbral sem pametni paket „Summernote“.

Summernote je pametni paket, namenjen pisanju obogatenega besedila. Z implementacijo Summernota sem na hiter in učinkovit način prišel do obogatenega besedila. Orodje sem vgradil v modalno pojavno okno, da je izgled bolj urejen. Prikaz orodja Summernote in modalnega okna je viden na sliki 4.13.



Slika 4.13: Orodje Summernote za urejanje obogatenega besedila.

Poglavje 5

Sklepne ugotovitve

Namen diplomskega dela je bil razvoj aplikacije, ki bi omogočala elektronski pregled in urejanje medicinskih algoritmov, obstoječih že več kot dvajset let samo v tiskani obliki.

Z implementacijo uporabniških zgodb nisem samo prenesel tiskane medicinske algoritme v elektronsko obliko, temveč sem zagotovil tudi možnost njihovega pregleda, urejanja tako strukturno kot podatkovno (možna je celo uporaba obogatenega besedila), dodajanje novih algoritmov in različen prikaz strani glede na uporabnikovo vlogo. Da bodo spremembe strokovno pregledane in potrjene, skrbi komisija za pregled, sprejemanje in zavračanje algoritmov.

Pri načrtovanju je bila ključnega pomena izbira programskega orodja za razvoj Algomeda. Z izborom ogrodja MeteorJS programskega okolja Node.js sem dosegel neodvisnost aplikacije od sistema. Dostop do Algomeda je mogoč preko računalnikov, tablic in pametnih mobilnih telefonov, operacijski sistem pa pri tem ne igra vloge, saj je aplikacija dostopna preko svetovnega spleta.

Dobro uporabniško izkušnjo in atraktiven izgled aplikacije sem zagotovil z uporabo knjižnice Twitter Bootstrap.

Čeprav vsebuje Bootstrap sloge za raznorazne komponente, kot so gumbi, spustni izbirni meniji ipd., se zavedam, da je izgled mogoče izboljšati. Glavna izboljšava bi bila razširitev sistema, ki trenutno podpira le prikaz, urejanje in

ustvarjanje binarnih dreves, na sistem, ki podpira uporabo dreves višjih stopenj. Naslednja možna izboljšava bi bila izbor še boljšega razvojega orodja. Veliko razvojnih ogrodij se bodisi približuje Facebookovima ogrodjema React (spletne aplikacije) in React Native (mobilne aplikacije) bodisi izvaja integracijo z Reactom – med drugim tudi Meteor. React je sicer precej kompleksnejši, je pa dandanes eno najhitrejših in najbolj optimiziranih razvojnih ogrodij.

Zaključujem, da je Algomed delujoč prototip aplikacije, ki ga je možno izboljševati in razviti do praktične uporabnosti – najmanj v slovenskem zdravstvenem prostoru.

Dodatek

Dodatek A

Izseki programske kode

Izsek programske kode A.1: Implementacija zbirke Customers in njene podatkovne sheme

```
// ZBIRKA
/* Definiramo novo zbirko in dolocimo ustrezne podatke za prikaz v
    administratorskem sistemu. Customers je od tu naprej v
    aplikaciji spremenljivka, ki se navezuje na zbirko Customers */
Customers = new orion.collection('customers', {
   /* neobvezni podatki, ki jih tu definiramo, kasneje pa lahko
       uporabimo */
  singularName: 'customer',
  pluralName: 'customers',
  /* title in link sta uporabljena v administratorskem sistemu za
      prikaz zbirke Customers */
  title: 'Customers',
  link: {
    title: 'Customers'
  },
   /* Tabular sluzi za prikaz zeljenih podatkov v administratorskem
       sistemu */
  tabular: {
    columns: [
```

```
      { data: "name", title: "Name"},
      { data: "surname", title: "Surname"},
      { data: "username", title: "Username"},
      { data: "email", title: "E-mail"},
      { data: "security_question", title: "Security question"},
      { data: "security_answer", title: "Security answer"},
      { data: "role", title: "Role", render: function(val, key,
          obj) {
          return UserRoles.findOne({_id: obj.role}).name;
        }
     },
      orion.attributeColumn('createdAt', 'createdAt', 'Created At')
    ]
  }
});

// SHEMA
/* Definiramo tudi shemo, torej polja in podatkovne tipe teh polj,
   lahko tudi relacije in nekatere druge moznosti, kot so optional
   (z moznostjo optional dolocimo, da je polje izbirno - pri
   kreaciji novega uporabnika, ga lahko dolocimo, lahko pa tudi ne
   */
Customers.attachSchema(new SimpleSchema({
  name: {
    type: String,
    label: "Name"
  },
  surname: {
    type: String,
    label: "Surname"
  },
  email: {
    type: String,
```

```
    label: "E-mail"
  },
  security_question: {
    type: String,
    label: "Security question"
  },
  security_answer: {
    type: String,
    label: "Security answer"
  },
  /* Polje role je z relacijo ena proti ena povezano s prej
     definirano zbirko UserRoles, ki prav tako kot spremenljivka
     Customers, ki kaze na zbirko Customers, kaze na zbirko
     UserRoles */
  role: orion.attribute('hasOne',
    { label: 'Role' },
    {
      collection: UserRoles,
      titleField: 'value',
      publicationName: 'userRoleCollection'
    }
  ),
  user_id: {
    type: String,
    label: "User ID"
  },
  createdAt: orion.attribute('createdAt')
}));
```

Izsek programske kode A.2: Primeri klicev nekaterih funkcij dodanega pametnega paketa accounts-password

```
// index.js
Accounts.forgotPassword({email: forgotten_email });  /* streznik
    poslje e-posto na naslov v spremenljivki forgotten_email, ki
    vsebuje povezavo do spletne strani za obnovitev gesla */
Accounts.resetPassword (
   Session.get('resetPasswordToken'), /* enolicno dolocen zeton, ki
       velja 15 minut in se doloci ob posiljanju e-poste */
   new_password, /* novo geslo, ki ga je vnesel uporabnik in je
       bilo validirano pred klicem te funkcije */
   function(error) {
      // koda, ki se izvede ob uspehu / napaki pri obnavljanju gesla
   }
});
```

Izsek programske kode A.3: Primer uporabe logičnih operatorjev in helperja `roleIsCommision` v HTML kodi

```
{{#if roleIsCommision}}
   <li>
      <a
         class="navbar-new"
         href="{{linkToPendingAlgorithms}}">
         Cakalna vrsta
      </a>
      <div id="notification">
         <p></p>
      </div>
   </li>
{{/if}}
```

Izsek programske kode A.4: Implementacija helperja `roleIsCommision`

```
...
Template.navbar.helpers({ /* Definicija vseh helperjev, ki jih
    uporablja predloga z imenom navbar */
   ...
   roleIsCommision: function() {
      if(Meteor.user()) {
         var id = Meteor.users.findOne(Meteor.userId())._id;
         var user = Customers.findOne({user_id: id});
         if(user && user.role)
         /* Nasli smo uporabnika, ki je trenutno prijavljen in ima
             definirano vlogo */
            return UserRoles.findOne({_id: user.role}).value == 2;
               /* Vrnemo True v primeru, ce je numericna vrednost
               uporabnikove vloge enaka 2, kar predstavlja
               komisijsko vlogo */
      }
      return false;
   },
   ...
});
...
```

Izsek programske kode A.5: Implementacija gumba za spremembo gesla v ospredju aplikacije in odziv na njegov pritisk v zaledju aplikacije

```
// manage-account.html
<button
   id="changePassword"
   type="button"
   class="btn btn-default">
   Spremeba gesla
</button>

// manage-account.js
// Na enak nacin kot helperje definiramo tudi dogodke
Template.manage_account.events({
   ...
   /* Uporabljajo se jQuery selektorji in sprejemljivi so vsi
   JavaScript dogodki. Format ukaza je:
   "[dogodek] [selektor]": function(...) {...} */
   "click #changePassword": function() {
      Modal.show("changePasswordModal");
   },
   ...
});
```

Izsek programske kode A.6: Implementacija migracije, označene z verzijo 1

```
Migrations.add({
  version: 1, // oznaka za verzijo
  name: "Initial data",
  up: function() {
    var ur1 = UserRoles.insert({
      value: 0,
      name: "pacient"
    });
    var ur2 = UserRoles.insert({
      value: 1,
      name: "zdravnik"
    });
    var ur3 = UserRoles.insert({
      value: 2,
      name: "kom. zdravnik"
    });
```

Literatura

[1] S. B. Mushlin and H. L. Greene II. *Decision Making in Medicine: An Algorithmic Approach, Third Edition*. Mosby Elsevier, Philadelphia, 2010

[2] R. D. Collins. *Algorithmic Diagnosis of Symptoms and Signs: A Cost–Effective Approach, Third Edition*. Wolters Kluwer, Lippincott Williams & Wilkins, Philadelphia, 2001.

[3] D. Keber in Z. Fras. *Zdravljenje notranjih bolezni*. Medicinski razgledi, Littera Picta, Ljubljana, Slovenija, 1992

[4] D. Keber in Z. Fras. *Razpoznavanje notranjih bolezni*. Medicinski razgledi, Littera Picta, Ljubljana, Slovenija, 1994

[5] Aplikacija, podobna Algomedu.
`http://www.medicalalgorithms.com/`.
[Online; Dostopano 18. 8. 2016]

[6] S. Tilkov and S. Vinoski. *Node. js: Using JavaScript to build high-performance network programs*. IEEE Internet Computing 14.6, pages 80–83, 2010.

[7] D. Turnbull. *7 Reasons to Develop Your Next Web App with Meteor*. SitePoint, July 2014.

[8] M. Lipuš. *Realno – časovne spletne aplikacije*. Fakulteta za elektrotehniko, računalništvo in informatiko, Univerza v Mariboru, Maribor, Slovenija, 2011

[9] V. Mahnič. *Tehnologija programske opreme*. Laboratorij za tehnologijo programske opreme, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, Ljubljana, Slovenija, 2011